Agilent 75000 SERIES C

Agilent E1488A
Instrument Memory Module

User's Manual

Agilent Technologies

Copyright© Agilent Technologies, Inc., 1995 - 2006

E1488-90003
E0806

Manual Part Number: E1488-90003

Printed: August 2006 Edition 2 Rev 2
Printed in Malaysia E0806

Contents

Warranty 5
WARNINGS 6
Safety Symbols 6
Declaration of Conformity 7
User's Notes 8

1. Getting Started

Chapter Contents 11
Preparation for Use 11
Installing the Memory Module 11
Device Driver 11
Memory Module Logical Address 13
Addressing the Memory Module (External Controller and PC) 13
Instrument and Programming Languages 14
SCPI Programming 14
SCPI Command Structure 14
C Language Example Programs 15
Agilent IBASIC/BASIC Language Programs 20
QuickBASIC Language Programs 23

2. Configuring the Memory Module

Chapter Contents 27
Warnings and Cautions 27
Setting the Logical Address 28
Setting the Module Configuration Switches 29
Enabling and Selecting the Backup Method 29
Configuring the Memory Size 29
Selecting the Interrupt Priority 32

3. Using the Memory Module

Chapter Contents 33
Module Description 33
Example Programs 33
Transferring Data Using GPIB 34
Storing Data 34
Outputting Data 35
Comments 38
Transferring Data Using The Local Bus 41
Storing Data to the Local Bus 41
Outputting Data to the Local Bus 45
Comments 48
Using Multiple Memory Modules on the Local Bus 49
Passing Data Through Memory Modules 49
Storing Data into Multiple Modules 54
Outputting Data from Multiple Modules 60
Comments 64

4. Understanding the Memory Module

Chapter Contents . . . 69
Local Bus Data Transfers . . . 69
Local Bus Description . . . 69
How to Transfer Data . . . 70
Local Bus Operating Modes . . . 71
The SCPI Status System . . . 73
Status System Overview . . . 73
The Standard Event Group . . . 74
The Operation Status Group . . . 75
The Questionable Data Group . . . 77
The Status Byte Group . . . 77

5. Command Reference

About this Chapter . . . 79
Command Types . . . 79
Common Command Format . . . 79
SCPI Command Format . . . 79
Compound Command Separator . . . 79
Abbreviated Commands . . . 80
Implied Commands . . . 80
Parameters . . . 80
Linking Commands . . . 82
SCPI Command Reference . . . 82
ABORt . . . 83
DISPlay . . . 84
:MONitor[:STATe] . . . 84
FORMat . . . 85
[:DATA] . . . 85
INITiate . . . 86
:CONTinuous . . . 86
[:IMMediate] . . . 87
MEMory . . . 88
:FREE? . . . 88
NONVolatile? . . . 88
STATus . . . 89
:OPC:INITiate . . . 89
:OPERation|QUEStionable:CONDition? . . . 90
:OPERation|QUEStionable:ENABle . . . 90
:OPERation|QUEStionable[:EVENt]? . . . 91
:OPERation|QUEStionable:NTRansition . . . 92
:OPERation|QUEStionable:PTRansition . . . 92
:PRESet . . . 93
SYSTem . . . 94
:ERRor? . . . 94
:VERSion? . . . 94
TRACe|DATA . . . 96
:CATalog? . . . 96
[:DATA] . . . 96
[:DATA]:COUNt . . . 97
[:DATA]:PARTial? . . . 98
:DEFine . . . 98
:DELete:ALL . . . 99

:DELete[:NAME] . . . 100
VINStrument . . . 101
[:CONFigure]:LBUS[:MODE] . . . 101
[:CONFigure]:LBUS:PAUSed? . . . 103
[:CONFigure]:LBUS:RESet . . . 104
[:CONFigure]:TEST:DATA . . . 104
[:CONFigure]:TEST:DATA? . . . 105
[:CONFigure]:VME[:MODE] . . . 106
[:CONFigure]:VME:RECeive:ADDRess:DATA? . . . 107
[:CONFigure]:VME:RECeive:ADDRess:READy? . . . 107
[:CONFigure]:VME:SEND:ADDRess:DATA? . . . 107
[:CONFigure]:VME:SEND:ADDRess:READy? . . . 108
:IDENtity? . . . 108
IEEE-488.2 Common Commands . . . 109
*CLS . . . 110
*DMC . . . 110
*EMC and *EMC? . . . 111
*ESE and *ESE? . . . 111
*ESR? . . . 112
*GMC? . . . 112
*IDN? . . . 112
*LMC? . . . 113
*LRN? . . . 113
*OPC . . . 114
*OPC? . . . 114
*PMC . . . 114
*RCL . . . 115
*RMC . . . 115
*RST . . . 116
*SAV . . . 116
*SRE and *SRE? . . . 117
*STB? . . . 117
*TST? . . . 118
*WAI . . . 118
Agilent E1488A Instrument Memory Module Command Quick Reference . . . 119
SCPI Conformance Information . . . 123

A. Specifications

Contents . . . 125
Instrument Specifications . . . 125
General VXI Characteristics . . . 126

B. Error Codes . . . 127

C. Register-Based Programming

Contents . . . 131
General Information . . . 131
Register Addressing . . . 131
The Base Address . . . 133
Memory Module Registers . . . 134
Register Assignments . . . 134
Register Descriptions . . . 135

Register-Based Example Program 148
System Configuration 148
Running the Program 148
Program Example 148

Certification

Agilent Technologies certifies that this product met its published specifications at the time of shipment from the factory. Agilent Technologies further certifies that its calibration measurements are traceable to the United States National Institute of Standards and Technology (formerly National Bureau of Standards), to the extent allowed by that organization's calibration facility, and to the calibration facilities of other International Standards Organization members.

Warranty

This Agilent Technologies product is warranted against defects in materials and workmanship for a period of one (1) year from date of shipment. Duration and conditions of warranty for this product may be superseded when the product is integrated into (becomes a part of) other Agilent products. During the warranty period, Agilent Technologies will, at its option, either repair or replace products which prove to be defective.

For warranty service or repair, this product must be returned to a service facility designated by Agilent Technologies. Buyer shall prepay shipping charges to Agilent and Agilent shall pay shipping charges to return the product to Buyer. However, Buyer shall pay all shipping charges, duties, and taxes for products returned to Agilent from another country.

Agilent warrants that its software and firmware designated by Agilent for use with a product will execute its programming instructions when properly installed on that product. Agilent does not warrant that the operation of the product, or software, or firmware will be uninterrupted or error free.

Limitation Of Warranty

The foregoing warranty shall not apply to defects resulting from improper or inadequate maintenance by Buyer, Buyer-supplied products or interfacing, unauthorized modification or misuse, operation outside of the environmental specifications for the product, or improper site preparation or maintenance.

The design and implementation of any circuit on this product is the sole responsibility of the Buyer. Agilent does not warrant the Buyer's circuitry or malfunctions of Agilent products that result from the Buyer's circuitry. In addition, Agilent does not warrant any damage that occurs as a result of the Buyer's circuit or any defects that result from Buyer-supplied products.

NO OTHER WARRANTY IS EXPRESSED OR IMPLIED. Agilent SPECIFICALLY DISCLAIMS THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

Exclusive Remedies

THE REMEDIES PROVIDED HEREIN ARE BUYER'S SOLE AND EXCLUSIVE REMEDIES. Agilent SHALL NOT BE LIABLE FOR ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES, WHETHER BASED ON CONTRACT, TORT, OR ANY OTHER LEGAL THEORY.

Notice

The information contained in this document is subject to change without notice. Agilent Technologies MAKES NO WARRANTY OF ANY KIND WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. Agilent shall not be liable for errors contained herein or for incidental or consequential damages in connection with the furnishing, performance or use of this material. This document contains proprietary information which is protected by copyright. All rights are reserved. No part of this document may be photocopied, reproduced, or translated to another language without the prior written consent of Agilent Technologies, Inc. Agilent assumes no responsibility for the use or reliability of its software on equipment that is not furnished by Agilent.

U.S. Government Restricted Rights

The Software and Documentation have been developed entirely at private expense. They are delivered and licensed as "commercial computer software" as defined in DFARS 252.227- 7013 (Oct 1988), DFARS 252.211-7015 (May 1991) or DFARS 252.227-7014 (Jun 1995), as a "commercial item" as defined in FAR 2.101(a), or as "Restricted computer software" as defined in FAR 52.227-19 (Jun 1987)(or any equivalent agency regulation or contract clause), whichever is applicable. You have only those rights provided for such Software and Documentation by the applicable FAR or DFARS clause or the Agilent standard software agreement for the product involved.

Agilent E1488A Instrument Memory Module User's Manual
Edition 2 Rev 2

Copyright © 1995-2006 Agilent Technologies, Inc. All Rights Reserved.

Printing History

The Printing History shown below lists all Editions and Updates of this manual and the printing date(s). The first printing of the manual is Edition 1. The Edition number increments by 1 whenever the manual is revised. Updates, which are issued between Editions, contain replacement pages to correct the current Edition of the manual. Updates are numbered sequentially starting with Update 1. When a new Edition is created, it contains all the Update information for the previous Edition. Each new Edition or Update also includes a revised copy of this printing history page. Many product updates or revisions do not require manual changes and, conversely, manual corrections may be done without accompanying product changes. Therefore, do not expect a one-to-one correspondence between product updates and manual updates.

Edition 1 . . . . . June 1993

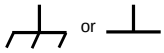
Edition 2 (Part Number E1488-90003). . . . . November 1995

Edition 2 Rev 2 (Part Number E1488-90003) . . . . . August 2006

Safety Symbols

Instruction manual symbol affixed to product. Indicates that the user must refer to the manual for specific WARNING or CAUTION information to avoid personal injury or damage to the product.

Indicates the field wiring terminal that must be connected to earth ground before operating the equipment—protects against electrical shock in case of fault.



Frame or chassis ground terminal—typically connects to the equipment's metal frame.

Alternating current (AC).

Direct current (DC).

Indicates hazardous voltages.

WARNING Calls attention to a procedure, practice, or condition that could cause bodily injury or death.

CAUTION Calls attention to a procedure, practice, or condition that could possibly cause damage to equipment or permanent loss of data.

WARNINGS

The following general safety precautions must be observed during all phases of operation, service, and repair of this product. Failure to comply with these precautions or with specific warnings elsewhere in this manual violates safety standards of design, manufacture, and intended use of the product. Agilent Technologies assumes no liability for the customer's failure to comply with these requirements.

Ground the equipment: For Safety Class 1 equipment (equipment having a protective earth terminal), an uninterruptible safety earth ground must be provided from the mains power source to the product input wiring terminals or supplied power cable.

DO NOT operate the product in an explosive atmosphere or in the presence of flammable gases or fumes.

For continued protection against fire, replace the line fuse(s) only with fuse(s) of the same voltage and current rating and type. DO NOT use repaired fuses or short-circuited fuse holders.

Keep away from live circuits: Operating personnel must not remove equipment covers or shields. Procedures involving the removal of covers or shields are for use by service-trained personnel only. Under certain conditions, dangerous voltages may exist even with the equipment switched off. To avoid dangerous electrical shock, DO NOT perform procedures involving cover or shield removal unless you are qualified to do so.

DO NOT operate damaged equipment: Whenever it is possible that the safety protection features built into this product have been impaired, either through physical damage, excessive moisture, or any other reason, REMOVE POWER and do not use the product until safe operation can be verified by service-trained personnel. If necessary, return the product to an Agilent Technologies Sales and Service Office for service and repair to ensure that safety features are maintained.

DO NOT service or adjust alone: Do not attempt internal service or adjustment unless another person, capable of rendering first aid and resuscitation, is present.

DO NOT substitute parts or modify equipment: Because of the danger of introducing additional hazards, do not install substitute parts or perform any unauthorized modification to the product. Return the product to an Agilent Technologies Sales and Service Office for service and repair to ensure that safety features are maintained.

Agilent Technologies	**DECLARATION OF CONFORMITY** According to ISO/IEC Guide 22 and CEN/CENELEC EN 45014	CE

Manufacturer's Name: Agilent Technologies, Incorporated
Manufacturer's Address: 815 – 14th St. SW
Loveland, Colorado 80537
USA

Declares, that the product

Product Name: Expandable Memory Card
Model Number: E1488A
Product Options: *This declaration covers all options of the above product(s).*

Conforms with the following European Directives:

The product herewith complies with the requirements of the Low Voltage Directive 73/23/EEC and the EMC Directive 89/336/EEC (including 93/68/EEC) and carries the CE Marking accordingly.

Conforms with the following product standards:

EMC	**Standard**	**Limit**
	CISPR 11:1990 / EN 55011:1991	*Group 1 Class A*
	IEC 801-2:1991 / EN 50082-1:1992	*4kV CD, 8kV AD*
	IEC 801-3:1984 / EN 50082-1:1992	*3 V/m*
	IEC 801-4:1988 / EN 50082-1:1992	*0.5kV signal lines, 1kV power lines*
	IEC 801-5 (Draft 1992)	3kV Low Energy, 1kV High Energy
	The product was tested in a typical configuration with Agilent Technologies or Hewlett-Packard Company test systems.	
Safety	*IEC 1010-1:1990+A1:1992 / EN 61010-1:1993* *Canada: CSA C22.2 No. 1010.1:1992* *UL 1244*	

3 May 2001
Date

Ray Corson
Product Regulations Program Manager

For further information, please contact your local Agilent Technologies sales office, agent or distributor.
Authorized EU-representative: Agilent Technologies Deutschland GmbH, Herrenberger Strabe 130, D 71034 Böblingen, Germany

Notes

Notes

Notes

Chapter 1
Getting Started

Chapter Contents

This chapter shows you how to install and begin using the Agilent E1488A Instrument Memory Module (known throughout this manual as the Memory Module). The main sections of this chapter are:

- Preparation for Use 11
  - Installing the Memory Module. 11
  - Device Driver 11
  - Memory Module Logical Address 13
  - Addressing the Memory Module 13
- Instrument and Programming Languages 14
  - C Language Example Programs. 15
  - Agilent IBASIC/BASIC Language Programs 20
  - QuickBASIC Language Programs 23

Preparation for Use

This section shows how to prepare the Memory Module for use. Included are instructions to install the module, download the Device Driver, and addressing the module.

Installing the Memory Module

The Memory Module can be installed in any C-Size VXIbus mainframe slot, except slot 0. For installation instructions in C-Size VXIbus Systems, refer to the C-size "Installation and Getting Started Guide".

Note For compliance with European EMC standards EN 55011 and EN 50082-1, backplane shields are included with each Agilent E1488A Instrument Memory Module ordered. The shields need only be installed in the VXIbus mainframe if it is necessary to comply with these standards.

Device Driver

The Device Driver allows the module to operate in an Agilent E1400 or E1401 Mainframe with an Agilent E1405 or E1406 Command Module using the Standard Commands for Programmable Instruments (i.e., SCPI). These drivers are supplied on the Memory Module's example program disks (see "Instrument and Programming Languages" on page 14 for more information on the disks). The Device drivers are called "MEMORY".

Installing the Drivers

For the procedure to download the drivers, refer to the Agilent E1405 or Agilent E1406 Command Module manual, or to the Driver Installation Note that came with the command module manual and the driver disks (the Agilent Part Number of the Installation Note is: E1401-90021 at the time this manual was published).

Command Module Requirements

The version number of the Agilent E1405 Command Module must be A.06.00 or above (use any version of the the Agilent E1406 Command Module). To determine the version number, send the IEEE 488.2 common command "*IDN?". A typical returned value (for Agilent E1405 Command Module) is:

HEWLETT-PACKARD,E1405B,0,A.06.00

where "A.06.00" is the version number.

Recommended Ways to Download the Drivers

Use one of the following ways to download the drivers.

Using BASIC with an Externally Connected GPIB Controller

This system downloads the driver over the General Purpose Interface Bus (GPIB[1]) with a BASIC controller. A typical controller is an HP Series 200/300 computer. Be sure to connect the computer to the Command Module's GPIB port. The program to download the driver is named "VXIDLD". It is located on the Agilent IBASIC driver/example program disk (Agilent part number E1488-10302). This disk is formatted in the LIF format.

Using Agilent IBASIC in an Agilent E1405 or Agilent E1406 Command Module

This system downloads the drivers using the Agilent IBASIC (i.e., Agilent Instrument BASIC) program language. This program language comes in an Agilent E1405 or Agilent 1406 Command Module with Option 020. IBASIC reads disks formatted in either the LIF or MS-DOS® format and thus can use either the LIF or DOS driver/example program disk (dependent on the disk drive used). The program to download the driver from the LIF disk (Agilent part number E1488-10302) is called "VXIDLD". The program to download the driver from the DOS disk (Agilent part number E1488-10301) is called "VXIDLD.GET".

1 GPIB is the implementation of the IEEE Standard 488.1-1978

Using and Externally Connected MS-DOS® Computer

This system downloads the driver using an MS-DOS computer connected to the Agilent E1405 or Agilent E1406 Command Module's RS-232 serial interface. The program to download the driver is named " VXIDLD.EXE" and is located on the DOS driver/example program disk (Agilent part number E1488-10301). This disk is formatted for MS-DOS.

Memory Module Logical Address

Each module in an C-Size system must have a unique Logical Address. For most purposes, use the Memory Module's default logical address of 24. If a different address is desired, go to " Addressing the Memory Module" in Chapter 2 on page 28 to change the setting.

Addressing the Memory Module (External Controller and PC)

To operate in a C-size system using GPIB, the modules in a system has a unique GPIB address. The GPIB address is a combination of the controller's interface select code, the Command Module's primary GPIB address, and the module's secondary GPIB address. A typical BASIC or IBASIC address (with a SCPI command is):

```
OUTPUT  70902;"*RST"
```

Interface Select Code (7): Determined by the address of the GPIB interface card in the controller. In most Agilent Technologies controllers, this card has a factory set address of 7, including the Agilent 82335 GPIB Interface Card (this card was used with an HP Vectra 386 compatible personal computer to create the C example programs).

Primary GPIB Address (**09**): This is the address of the GPIB port on the Command Module. Valid addresses are 0 to 30. The Command Module has a factory set address of 9.

Secondary GPIB Address (02): This address is derived from the logical address of the device (the Memory Module) by dividing the logical address by 8. Thus, for the E1488A Memory Module factory set logical address of 16, the secondary address is 02.

Using an Embedded Controller

As a register-based device, the E1488A Memory Module can be programmed over the VXIbus backplane from an Agilent E1499A V/382 or other embedded controller at the register level. For register programming, see Appendix C of the manual.

Instrument and Programming Languages

The Agilent E1488A Memory Module uses SCPI (Standard Commands for Programmable Instruments) as the programming language. Chapter 3, "Command Reference" documents the the applicable SCPI commands. Chapter 2, "Using the Memory Module" shows how to program the module using the commands.

The programs shown in the manual are SCPI programs written in C and are on the following which ships with the manual:

- **Agilent E1488A Device Driver: C Language - 3.5" 1.44 Mbyte disk (E1488-10301)**
- **Agilent E1488A Device Driver: BASIC/IBASIC Language - 3.5" 720 kbyte disk (E1488-10302)**

SCPI Programming

SCPI (Standard Commands for Programmable Instruments) is an ASCII-based instrument command language designed for test and measurement instruments. The Agilent E1405 or Agilent E1406 Command Module with the Memory Module's Device Driver installed interprets the ASCII command strings and sets the module to the correct states. The Command Module does this by setting the Memory Module registers to the appropriate states.

SCPI Command Structure

A SCPI command structure is a hierarchal structure that usually consists of a top level (or root) command, one or more lower level commands, and their parameters. The following example shows part of a typical subsystem:

```
[TRACe | DATA]
    :DELete
        :ALL
        [:NAME]  <name>
```

TRACe or DATA is the root keyword of the command, :DELete is the second level keyword, and :ALL and :NAME are the third level keyword. A colon (:) always separates a command keyword from a lower level keyword as shown below:

```
TRACe:DELete:ALL
```

A semicolon (;) separates two commands within the same subsystem. For example, this command message:

```
TRACE:DELete:NAME  <name>;:ALL
```

Is the same as these two commands:

```
TRACE:DELete:NAME <name>
TRACE:DELete:ALL
```

SCPI Command Format

The typical format of commands listed in the command reference and throughout this manual is:

```
FORMat[:DATA] <format>,[<length>]
```

Command headers enclosed in brackets are optional. Required are the UPPER CASE letters in the header; you can omit the lower case letters. **To aid in learning the command sets, all command headers are included in the example programs; however, the command headers are abbreviated**. In the example programs, the previous command with an *type* parameter of ASCii for [:DATA] appears as:

```
FORM:DATA ASC
```

C Language Example Programs

All of the C Language example programs in this manual are written for the Agilent 82335 GPIB Interface Card using the GPIB Command Library for C. Unless otherwise noted, the library functions used in the programs are compatible with the ANSI C standard.

The following identifies the system on which the programs are written, shows how to compile the programs, and gives a typical example program.

System Configuration

The C programs were developed on the following system:

Controller:	HP Vectra 386/25 personal computer (386 processor operated at 25 MHz)
GPIB Interface Card:	Agilent 82335 GPIB Interface with Command Library
Mainframe:	Agilent 75000 Series C
Slot0/Resource Manager:	Agilent E1405B Command Module
Agilent E1488A Logical Address:	16
Instrument Language:	SCPI

C Compiler Used

Unless otherwise noted, all C Language programs were compiled (and tested) using the following compiler:

Borland Turbo C++© Version 3.1

Running a Program

To run a program, first compile and link the program to make an executable file. To compile a program:

- Link the appropriate GPIB C library (located on the GPIB Command Library disk that came with the GPIB Interface Card). Use the following libraries:
  - **Microsoft® C and QuickC©:** clhpib.lib
  - **Turbo C© and C++©:** tchhpib.lib
- If NOT compiling in the Large/Huge memory model, include the "cfunc.h" header file in the program (located on the GPIB Command Library disk that came with the GPIB Interface Card).
- Be sure the necessary paths have been added to the AUTOEXEC.BAT file for the compilers to find the library and header files (see the appropriate C Language manual to set the proper paths).

To compile the programs from the DOS command line using the Large memory model, execute the following:

- **Microsoft® C:**

  cl /AL <*path* \program name> *path* \clhpib.lib

- **Microsoft® QuickC©:**

  qcl /AL <*path* \program name> *path* \clhpib.lib

- **Turbo C© and Turbo C++©:**

  tcc -ml <*path* \program name> *path* \tchhpib.lib

Change the "/AL" and "-ml" parameters to the appropriate types when compiling in the smaller memory models (see the appropriate C Language manual for the parameter type).

C Program Example

The program that follows is an example program written in C using the Agilent 82335 GPIB Interface Card. The program stores data into memory from the Local Bus. Included is also an error checking function to make sure the module commands were sent correctly.

```
/* lb_str.c - This program:                                                    */
/*          1.) Calls function to setup user module to output data (user       */
/*              writes and supplies the function)                              */
/*          2.) Sends setup commands to store 2097152 bytes into memory        */
/*              from the Local Bus                                             */
/*          3.) Sends commands to initiate the Memory Module to read data      */
/*              from the Local Bus                                             */
/*          4.) Calls function to initiate user module to output the data      */
/*              to the Local Bus (user writes and supplies the function)       */
/*                                                                             */
/*   The program assumes a Logical Address of 16 (GPIB Secondary Address of    */
/*   02) for the Memory Module                                                 */

        /* Include the following header files */
#include <stdio.h>
#include <string.h>
#include <cfunc.h>          /* This file is from the GPIB Command Library */

#define ADDR 70902L     /* Assign an I/O path between the computer and Memory Module */

        /* Functions */
void store_data(void);
void send_data(void);
void cmd_exe(char *commands[], int length);
void rst_clr(void);
void check_error(char *func_tion);

/*****************************************************************************/
void main(void)
        /* Main function that runs the program */
{
        rst_clr();          /* Reset the Memory Module */
        store_data();  /* Store Local Bus Data into memory */
}
/*****************************************************************************/
void store_data(void)
        /* This function stores data from the Local Bus into memory */
{
        static char *set_commands[] = /* Use “set_commands” to setup the Memory Module */
        {
        “VINS:CONF:VME:MODE  OFF”,      /* Turns register-level data transfer off */

        “TRAC:DEL:ALL”,                 /* Deletes trace from memory */
        “TRAC:DEF  lb_data, 2097152",   /* Defines trace name and size */
```

Continued on Next Page

```
        "VINS:CONF:LBUS:RES",           /* Resets Local Bus circuitry */
        "VINS:CONF:LBUS:MOD   CONS",  /* Sets Local Bus Mode to CONSume */

        "STAT:OPC:INIT  OFF"            /* Execute *OPC? after INIT is parsed */
        },

        *init_commands[] =          /* Use "init_commands" to start storing data */
        {
        "INIT:IMM",                     /* Initiate the Memory Module */
        "*OPC?"                         /* Wait for INIT to complete before continuing */
        };

        float      rdy;

           /* Setup user module to output Local Bus data, but do not
              initiate the module to output the data */
/* USER DEFINED FUNCTION */

           /* Execute the module's setup commands */
        cmd_exe(set_commands, sizeof(set_commands) / sizeof(char*));

           /* Call function to check for errors */
        check_error("store_data (set_commands)");

           /* Execute the module's INIT and *OPC? commands */
        cmd_exe(init_commands, sizeof(init_commands) / sizeof(char*));

           /* Enter *OPC? response; continue when module is ready */
        IOENTER(ADDR, &rdy);

           /* Initiate user module to output data to the Local Bus;
              the module must be in the slot next to and the left of the
              Memory Module */
/* USER DEFINED FUNCTION */

           /* Call function to check for errors */
        check_error("store_data (data stored)");
}
/*****************************************************************************/
void cmd_exe(char *commands[], int length)
        /* Function that sends commands to the module */
{
        intloop;
```

Continued on Next Page

```
        /* Execute each command group using a loop */
    for (loop = 0; loop < length; loop++)
        IOOUTPUTS(ADDR, commands[loop], strlen(commands[loop]));
}
/*****************************************************************************/
void rst_clr(void)
    /* Function that resets the module and clears the error register */
{
    float    into;

    IOOUTPUTS(ADDR, "*RST;*OPC?", 10);
    IOENTER(ADDR, &into);
    IOOUTPUTS(ADDR, "*CLS;*OPC?", 10);
    IOENTER(ADDR, &into);
}
/*****************************************************************************/
void check_error(char *func_tion)
    /* Function that checks for module generated errors */
{
    char   into[161];
    int    length = 160;

    IOOUTPUTS(ADDR, "SYST:ERR?", 9);          /* Query error register */
    IOENTERS(ADDR, into, &length);            /* Enter error message */

    if (atoi(into) != 0)                       /* Determine if error is present */
                                               /* If errors present, print and exit */
    {
        while (atoi(into))
        {
            printf("Error %s (in function %s)\n\n", into, func_tion);
            length = 160;
            IOOUTPUTS(ADDR, "SYST:ERR?", 9);
            IOENTERS(ADDR, into, &length);
        }

        exit(1);
    }
}
```

Agilent IBASIC/BASIC Language Programs

The following information identifies the system on which the Agilent IBASIC/BASIC program examples were written, and also shows a typical Agilent IBASIC/BASIC program.

System Configuration

Except where noted, the program example in Agilent IBASIC was developed on the following system:

Mainframe: Agilent 75000 Series C

Slot 0/Resource Manager: Agilent E1405B Command Module (with IBASIC option)

Agilent E1488A Logical Address: 16

Instrument Language: SCPI

Loading Programs

The Agilent IBASIC program was saved on the driver disk as an ASC file. The program name is shown in the program title and in line 1 of the program (i.e., 1 !RE_SAVE"LB_STR"). To load a program, type:

GET " file name" (i.e. GET "LB_STR")

Changing IBASIC Programs to BASIC Programs

You can use any IBASIC program saved as an ASCII file on a LIF formatted disk in an BASIC controller. Simply change the GPIB address from an IBASIC address to a BASIC address.

For example, the IBASIC address for the example program is:

"80902."

where "8" is the select code, "09" is the primary GPIB address, and "02" is the secondary GPIB adderss. Since most BASIC GPIB controllers have a select code of "7", change the address to:

"70902"

(Be sure to delete the decimal point at the end of the address; IBASIC places the decimal point there.)

IBASIC Program Example

The program that follows is an example program written in IBASIC. The program stores data into memory from the Local Bus. Included is also an error checking function to make sure the module commands were sent correctly.

```
1   ! RE-SAVE"LB_STR"
2   ! This program:
3   !       1.) Calls subprogram to setup user module to output data (user
4   !           writes and supplies the subprogram
5   !       2.) Sends setup commands to store 2097152 bytes into memory
6   !           from the Local Bus
7   !       3.) Sends commands to initiate the Memory Module to read data
8   !           from the Local Bus
9   !       4.) Calls routine to initiate user module to output the data
10  !           to the Local Bus (user writes and supplies the routine)
11  !
12  !The program assumes a Logical Address of 16 (GPIB Secondary Address of
13  !02) for the Memory Module
14  !
20  ! Assign I/O path between Agilent IBASIC and the Memory Module.
30  ASSIGN @Mem TO 80902.
40  COM @Mem
50  !
60  ! Call the subprogram which resets the Memory Module.
70  CALL Rst
80  !
90  ! Call subprogram to setup user module to output Local Bus data,
100   ! but do not initiate the module to output the data
110   ! USER DEFINED PROGRAM
120   !
130   ! Call subprogram to setup Memory Module
140   CALL Setup_mod
150   !
160   ! Call subprogram to check for errors
170   CALL Errmsg
180   !
190   ! Call subprogram to inititate Memory Module to receive data
200   CALL Init_mod
210   !
220   ! Call subprogram to initiate user module to output data to the Local
230   ! Bus; the module must be in the slot immediately to the left of
240   ! the Memory Module
250   ! USER DEFINED PROGRAM
260   !
270   ! Call subprogram to check for errors
280   CALL Errmsg
290   !
300   END
310   !
320   !
```

Continued on Next Page

```
330   SUB Setup_mod
340 Setup_mod: ! Setup the Memory Module
350     COM @Mem
360     OUTPUT @Mem;"VINS:CONF:VME:MODE  OFF" ! Turns register-level data transfer
off
370   !
380     OUTPUT @Mem;"TRAC:DEL:ALL" ! Deletes trace from memory
390     OUTPUT @Mem;"TRAC:DEF  lb_data,2097152" ! Defines trace name and size
400   !
410     OUTPUT @Mem;"VINS:CONF:LBUS:RES" ! Resets Local Bus circuitry
420     OUTPUT @Mem;"VINS:CONF:LBUS:MODE  CONS" ! Sets Local Bus Mode to
CONSume
430   !
440     OUTPUT @Mem;"STAT:OPC:INIT OFF" ! Execute *OPC? after INIT is parsed
450   SUBEND
460   !
470   SUB Init_mod
480 Init_mod:  ! Execute the module's INIT and *OPC? commands
490     COM @Mem
500     OUTPUT @Mem;"INIT" ! Inititate the memory Module
510     OUTPUT @Mem;"*OPC?" ! Wait for INIT to complete before continuing
520     ENTER @Mem;Complete ! Enter *OPC? respone; continue when module is ready
530   SUBEND
540   !
550   SUB Rst
560 Rst:  ! Subprogram which resets the Memory Module
570     COM @Mem
580     OUTPUT @Mem;"*RST;*OPC?"   ! Reset the module
590     ENTER @Mem;Complete
600     OUTPUT @Mem;"*CLS;*OPC?"   ! Clear the module
610     ENTER @Mem;Complete
620   SUBEND
630   !
640   SUB Errmsg
650 Errmsg:  ! Subprogram which displays module's programming errors
660     COM @Mem
670     DIM Message$[256]
680     REPEAT
690       OUTPUT @Mem;"SYST:ERR?" ! read module's error queue
700       ENTER @Mem;Code,Message$
710       IF Code<>0 THEN
723         PRINT Code,Message$
730       END IF
740     UNTIL Code=0
750   SUBEND
```

QuickBASIC Language Programs

The following information identifies the system on which the QuickBASIC program examples were written, and also shows a typical QuickBASIC program.

System Configuration

The QuickBASIC program was developed on the following system:

Controller: HP Vectra 386/25 personal computer (386 processor operated at 25 MHz)

GPIB Interface Card: Agilent 82335 GPIB Interface with Command Library

Mainframe: Agilent 75000 Series C

Slot0/Resource Manager: Agilent E1405B Command Module

Agilent E1488A Logical Address: 16

Instrument Language: SCPI

Running a Program

To run a program, first compile and link the program to make an executable file. You can either compile and link the program in the QuickBASIC environment, or compile and link separately on the command line, explained as follows.

Using the QuickBASIC Environment

To compile and link a program, select the QuickBASIC environment as follows:

qb <*path* \program name> /l *path* \qbhpib

This loads both the program and GPIB library into the QuickBASIC environment. Note that this only selects the QuickBASIC version 4.0 and above environment. For other versions of QuickBASIC, refer to the "Agilent 82335 Using the GPIB Interface and Command Library" manual.

Using the Command Line

To compile and link using the command line is a two step method as follows:

First, compile the program by executing:

bc <*path* \program name>

This generates an object listing that has the same name as the program name with an OBJ extension. This object file must be linked to the GPIB library.

Second, link the GPIB library:

link <*path* \program name>

After prompted for a run file, press the Enter key. The executable file with an EXE extension is then generated.

QuickBASIC Program Example

The program that follows is an example program written in QuickBASIC. The program stores data into memory from the Local Bus. Included is also an error checking function to make sure the module commands were sent correctly.

```
' LB_STR.BAS -    This program:
'                 1.) Calls routine to setup user module to output data (user
'                     writes and supplies the routine)
'                 2.) Sends setup commands to store 2097152 bytes into memory
'                     from the Local Bus
'                 3.) Sends commands to initiate the Memory Module to read data
'                     from the Local Bus
'                 4.) Calls routine to initiate user module to output the data
'                     to the Local Bus (user writes and supplies the routine)
'
'  The program assumes a Logical Address of 16 (GPIB Secondary Address of
'  02) for the Memory Module

DECLARE SUB StoreData ()
DECLARE SUB CmdExe (Commands$(), Length%)
DECLARE SUB RstClr ()
DECLARE SUB CheckError (FuncType$)

COMMON SHARED ADDR&

REM $INCLUDE: 'QBSETUP'

ADDR& = 70902   'Assign an I/O path between the computer and the Memory Module

CALL RstClr  ' Reset the Memory Module
CALL StoreData  ' Store Local Bus Data into memory

END
```

Continued on Next Page

```
SUB CheckError (FuncType$)
      ' Routine that checks for module generated errors

Max.Length% = 160: Actual.Length% = 0
Info$ = SPACE$(Max.Length%)

CALL IOOUTPUTS(ADDR&, "SYST:ERR?", 9)  ' Query error register
CALL IOENTERS(ADDR&, Info$, Max.Length%, Actual.Length%)
Info$ = LEFT$(Info$, Actual.Length%)

IF VAL(Info$) <> 0 THEN
      CLS
      WHILE VAL(Info$) <> 0
         PRINT : PRINT Info$; "  in routine: "; FuncType$
         Actual.Length% = 0
         CALL IOOUTPUTS(ADDR&, "SYST:ERR?", 9)
         CALL IOENTERS(ADDR&, Info$, Max.Length%, Actual.Length%)
         Info$ = LEFT$(Info$, Actual.Length%)
      WEND
      END
END IF

END SUB

SUB CmdExe (Commands$(), Length%)
      ' Routine that sends commands to the module

FOR I = 1 TO Length%
      CALL IOOUTPUTS(ADDR&, Commands$(I), LEN(Commands$(I)))
NEXT I

END SUB

SUB RstClr
      ' Routine that resets the module and clears the error register

CALL IOOUTPUTS(ADDR&, "*RST;*OPC?", 10)
CALL IOENTER(ADDR&, Into!)

CALL IOOUTPUTS(ADDR&, "*CLS;*OPC?", 10)
CALL IOENTER(ADDR&, Into!)

END SUB
```

Continued on Next Page

```
SUB StoreData
        ' This routine stores data from the Local Bus into memory

DIM SetCommands$(1 TO 6)  ' Use "SetCommands$" to setup the memory Module
DIM InitCommands$(1 TO 2)  ' Use "InitCommands$" to start storing data

SetCommands$(1) = "VINS:CONF:VME:MODE OFF"  ' Turns register-level data transfer off

SetCommands$(2) = "TRAC:DEL:ALL"  ' Deletes trace from memory
SetCommands$(3) = "TRAC:DEF lb_data,2097152"  ' Defines trace name and size

SetCommands$(4) = "VINS:CONF:LBUS:RES"  ' Resets Local Bus circuitry
SetCommands$(5) = "VINS:CONF:LBUS:MODE CONS"  ' Sets Local Bus mode to CONSume

SetCommands$(6) = "STAT:OPC:INIT OFF"  ' Execute *OPC? after INIT is parsed

InitCommands$(1) = "INIT:IMM"  ' Initiate the Memory Module
InitCommands$(2) = "*OPC?"  ' Wait for INIT to complete before continuing

        ' Setup user module to output Local Bus data, but do not
        ' initiate the module to output the data
' USER DEFINED ROUTINE

        ' Execute the module's setup commands
CALL CmdExe(SetCommands$(), 6)

        ' Call routine to check for errors
CALL CheckError("StoreData (SetCommands)")

        ' Execute the module's INIT and *OPC? commands
CALL CmdExe(InitCommands$(), 2)

        ' Enter *OPC? response; continue when module is ready
CALL IOENTER(ADDR&, Into!)

        ' Initiate user module to output data to the Local Bus;
        ' the module must be in the slot immediately to the left of
        ' the Memory Module
' USER DEFINED ROUTINE

        ' Call routine to check for errors
CALL CheckError("StoreData (data stored)")

END SUB
```

Chapter 2
Configuring the Memory Module

Chapter Contents

This chapter shows you how to set the Agilent E1488A Instrument Memory Module's (Memory Module's) Logical Address, select the backup method, select the interrupt priority, an configure the module for additional memory. The main sections of this chapter are:

- Warnings and Cautions 27
- Setting the Logical Address 28
- Setting the Module Configuration Switches 29
  - Enabling and Selecting the Backup Method 29
  - Configuring the Memory Size 29
- Selecting the Interrupt Priority. 32

Warnings and Cautions

Warning **SHOCK HAZARD. Only service-trained personnel who are aware of the hazards involved should install, remove, or configure the Memory Module. Before installing any module, disconnect AC power from the mainframe.**

Caution **STATIC ELECTRICITY. Static electricity is a major cause of component failure. To prevent damage to the electrical components in the Memory Module, observe anti-static techniques whenever working on a module.**

Setting the Logical Address

The Memory Module's default Logical Address is 16. If other modules in your mainframe has the same address as the Memory Module, set the Logical Address of the other or the Memory Module to a different value. If the mainframe uses the Agilent E1405B/E1406A Command Module, be sure to set the Logical Address to a value that is a multiple of 8.

Figure 2-1 shows the default Logical Address (16) of the Memory Module. Use the figure to change the setting to a different value.

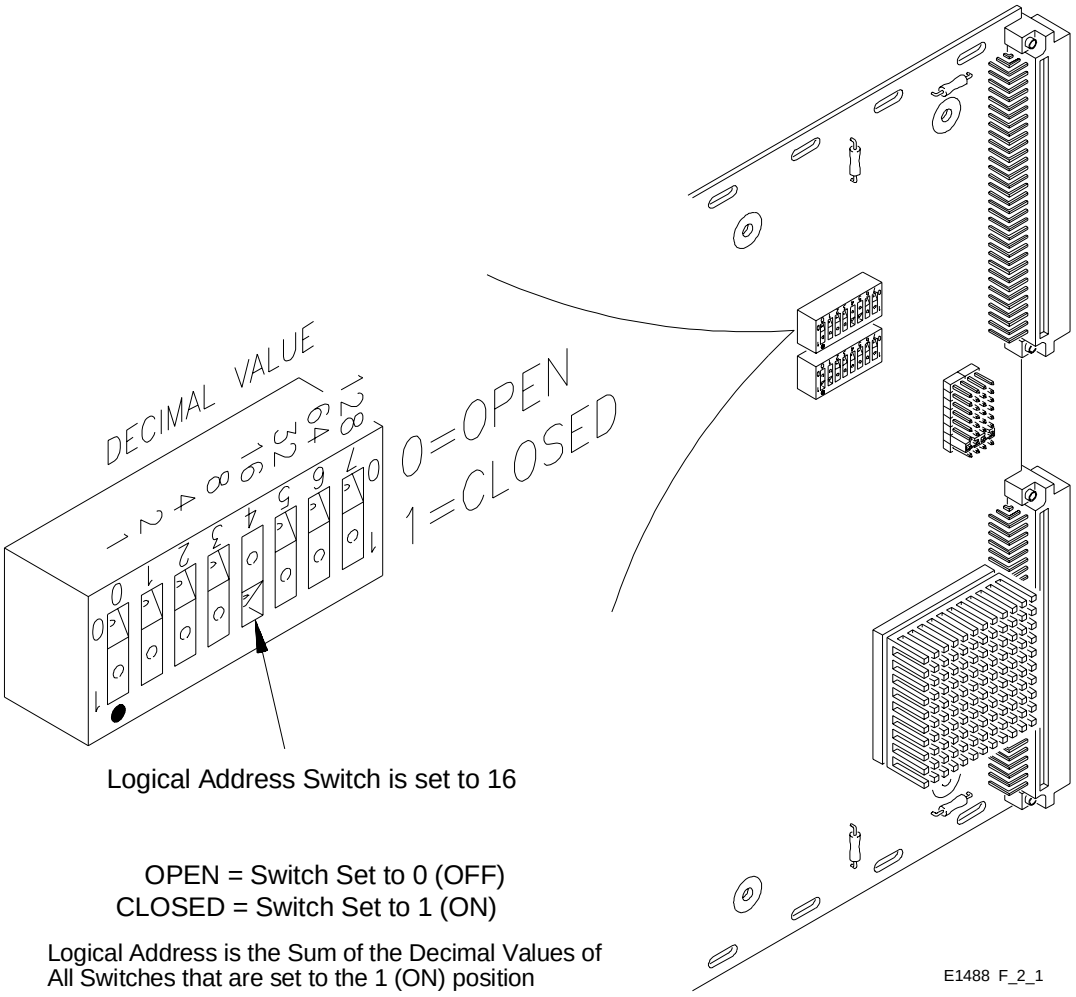

DECIMAL VALUE
0=OPEN
1=CLOSED
Logical Address Switch is set to 16
OPEN = Switch Set to 0 (OFF)
CLOSED = Switch Set to 1 (ON)
Logical Address is the Sum of the Decimal Values of
All Switches that are set to the 1 (ON) position
E1488 F_2_1

Figure 2-1. Logical Address

Setting the Module Configuration Switches

Figure 2-2 shows the location of the configuration switches. These switches are used to enable memory backup and configure the module for additional memory (i.e., any memory above 2 Mbytes).

Caution **Keep switch #3 in the 0 (OFF) position (as shown in Figure 2-2) or the Memory Module will operate incorrectly.**

Enabling and Selecting the Backup Method

The Memory Module has a feature that allow you to backup the data that is in memory (i.e., keep data in memory), whenever you disconnect the module's power source. To enable the backup, first set switch 7 on the configuration switch to the 1 (ON) position. Then select the backup method using the jumpers shown in Figure 2-2. The different backup methods are:

- Front Panel BNC Connector (allowable voltage range is from +3.5Vdc to +4.5 Vdc--see Caution below)
- "VSTBY" on the Backplane (allowable voltage range is from +3.5Vdc to +4.5 Vdc--see Caution below)
- Internal Battery

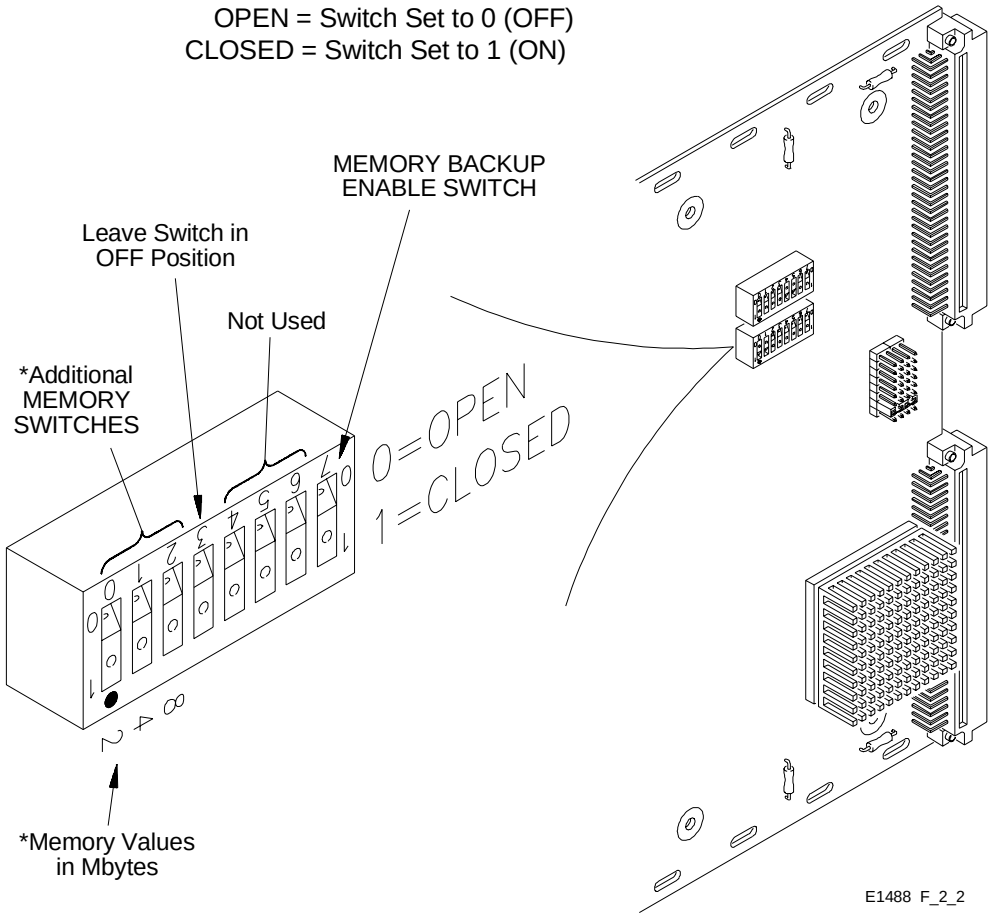
Caution **The maximum voltage the can be applied for backup is +4.5Vdc. Exceeding this voltage will damage the module.**

In most cases, you should use the Front Panel BNC Connector instead of the the backplane "+5VSTBY" line. You should use the "+5VSTBY" line only when the Agilent E1488A is the only module using that line. This is because other types of VXI modules generally require greater than 4.5Vdc.

Configuring the Memory Size

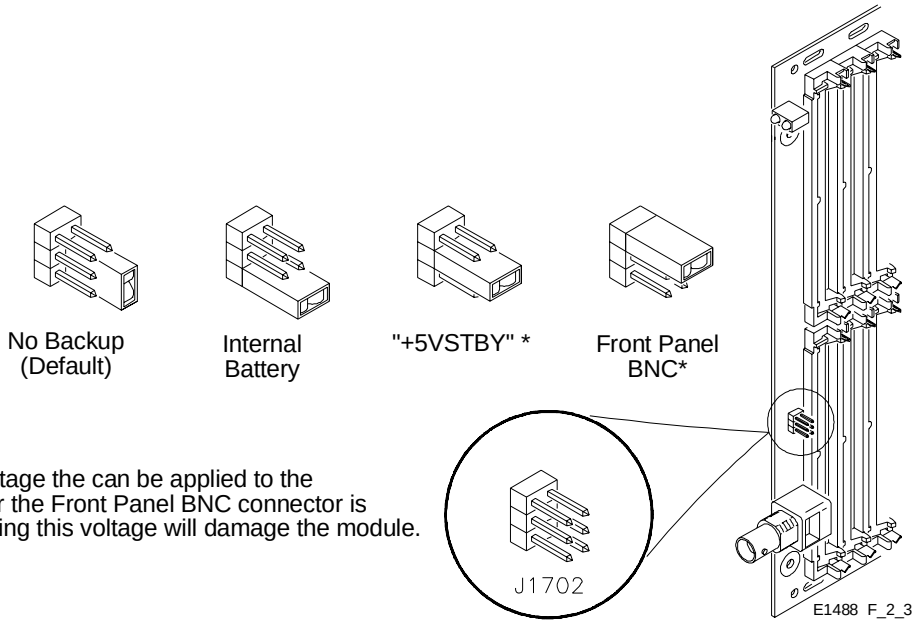
The standard memory size of the Memory Module is 2 Mbytes. You can increase this value up to 16 Mbytes in 2 Mbytes steps. Any memory added above the standard 2 Mbyte value is considered additional memory.

Normally there is no need to configure the module for the correct memory size. The module should be configured at the factory for the correct memory size. However, if additional memory was added at a later time, the memory configuration switches must be in the correct position for the module to recognize the additional memory. Figure 2-3 shows the location of the switches and the setting for 2 Mbyte of memory. If needed, use the figure to determine if the module is configured for any additional memory.



* Determine additional memory by the sum of the values of the switches set to 1 (ON). For example, if the additional memory is 10 Mbytes (for a total of 12 Mbytes memory), switches 0 and 2 are set to 1 (OFF) (i.e., 2 + 8 = 10 Mbytes additional memory).

Figure 2-2. Memory Module Configuration Switches



***Caution**

The maximum voltage the can be applied to the "+5VSTBY" line or the Front Panel BNC connector is +4.5Vdc. Exceeding this voltage will damage the module.

Figure 2-3. Selecting the Backup Method

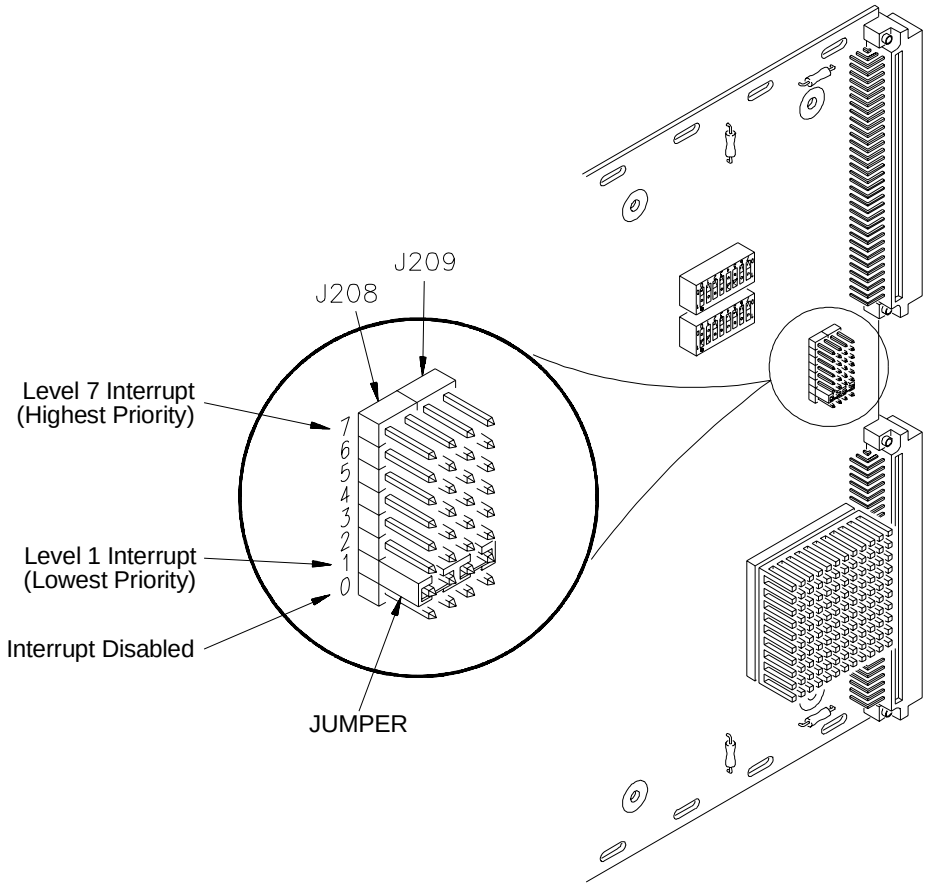


Figure 2-4. Selecting the Interrupt Priority

Selecting the Interrupt Priority

The Memory Module can generate interrupts when the IRQ Register is enabled (see Appendix C, "Register Programming" for more information). You can either disable the interrupts or select seven different interrupt priority levels for the module generated interrupts. The lowest level is level 1 and the highest is Level 7. Use Figure 2-4 to disable the interrupt capability, or change the interrupt priority (default value is 1).

Chapter 3
Using the Memory Module

Chapter Contents

This chapter shows how to use the Agilent E1488A Instrument Memory Module (Memory Module) using the SCPI instrumentation language. The sections are as follows:

- Module Description 33
- Example Programs 33
- Transferring Data Using GPIB 34
  - Storing Data 34
  - Outputting Data 35
- Transferring Data Using The Local Bus 41
  - Storing Data from the Local Bus 41
  - Outputting Data to the Local Bus 45
- Using Multiple Memory Modules on the Local Bus 49
  - Passing Data Through Memory Modules 49
  - Storing Data into Multiple Modules 54
  - Outputting Data from Multiple Modules 60

Module Description

The Agilent E1488A is a C-Size register-based VXIbus Module used to store and retrieve instrument data at a fast transfer rate. Dependent on the option ordered, the module's memory size can be from 2 Mbytes to 16 Mbytes.

The Agilent E1488A Memory Module is optimized to send or receive data on the VXIbus Local Bus at high rates of speed (>80Mbytes/sec). Data may also be retrieved from or stored to the Memory Module via the VME data transfer bus.

You can use single or multiple Memory Modules to store or output data on the Local Bus. The modules can also pass data through to other modules.

Example Programs

Most example programs in this chapter are very similar to the C Example Program in Chapter 1 of this manual. Thus, this chapter lists only those functions that are different from the functions of the program in Chapter 1.

Transferring Data Using GPIB

This program shows how to store (i.e., receive) and output (i.e., send) data to and from the Agilent E1488 Memory Module. This program uses the TRACE | DATA[:DATA] command to transfer the data. The command transfers data when an external GPIB controller is used with a command module.

Use the above command to transfer data from/to other modules that do not have Local Bus capabilities. This is the slower method for data transfer.

Storing Data

The following commands configure the module to store (receive) data from an external controller.

1. **Disable the Register-level VME Bus Data Transfer Mode**
   VINStrument[:CONFigure]:VME[:MODE] OFF

   This command disables the register-level data transfer on VMEbus. (No need to send this command if the module was previously reset.)

2. **Delete the Trace Currently in Memory**
   TRACE | DATA:DELete:ALL

   This command deletes the trace currently in memory (see next step) to allow for new data to be stored.

3. **Select the Trace Name and Memory Size for Data Storage**
   TRACE | DATA:DEFine <*name*>,<*length*>

   This command selects the trace name under which the data is to be stored and also sets the memory size (i.e., maximum number of data bytes to be stored).

4. **Store the Data into Memory**
   TRACE | DATA[:DATA] <*name*>,<*data_list*>

   Store the data into memory into the location defined by the TRACE | DATE:DEFine <*name*>,<*length*> command.

Outputting Data

The following commands configure the module to output (send) data to the external controller.

1. **Set the Returned Data Format**
   FORMat[:DATA] PACK

   This command sets the returned data format to PACKed (for faster data transfer). Default value is ASCii (slower data transfer).

2. **Determine Amount of Data Currently Stored (Optional Step)**
   TRACE | DATA[:DATA]:COUNt?

   This command returns the number of bytes currently stored by the memory module.

3. **Read and Return Some Data Bytes from Memory**
   TRACE | DATA[:DATA]:PARTial? <*name*>,1,4096

   This command returns 4096 bytes from memory. To return all the data, see "Returning All Data that is in Memory" on page 39.

C Program Example (dat_xfr.c)

This program randomly generates 1 Mbytes of data, stores it into memory, then reads 4096 bytes, and stores those into an array.

```
/* dat_xfr.c -  This program:                                          */
/*              1.) Transfers 1048576 bytes of data to the Memory Module in   */
/*                  64 blocks of 16384 bytes each                     */
/*              2.) Reads 8192 bytes of data back from the Memory Module      */
/*                                                                    */
/*  The program assumes a Logical Address of 16 (GPIB Secondary Address of   */
/*  02) for the Memory Module                                         */

      /* Include the following header files */
#include <stdio.h>
#include <string.h>
#include <malloc.h>
#include <stdlib.h>
#include <cfunc.h>     /* This file is from the GPIB Command Library */

#define ISC 7L              /* Assigns the GPIB select code */
#define ADDR 70902L    /* Assign an I/O path between the computer and Memory Module */
```

Continued on Next Page

```
        /* Functions */
void gen_data(void);
void read_data(void);
void cmd_exe(char *commands[], int length);
void rst_clr(void);
void check_error(char *func_tion);

/*****************************************************************************/
void main(void)
        /* Runs the program */
{
        rst_clr();          /* Reset the Memory Module */
        gen_data();     /* Send data to the Memory Module */
        read_data();    /* Reads data from the Memory Module */
}
/*****************************************************************************/
void gen_data(void)
        /* Functions that generates and sends the data to the module */
{
        static char *set_commands[] =   /* Use “set_commands” to setup the module */
        {
        “VINS:CONF:VME:MODE  OFF”,       /* Turns register-level data transfer off */

        “TRAC:DEL:ALL”,                          /* Deletes trace from memory */
        “TRAC:DEF  norm_data, 1048576"   /* Defines trace name and size */
        },

        *out_command =

        “TRACE:DATA  norm_data, #71048576";      /* Command to store data into memory */

        unsigned char   *gen_data;
        int          data_size = 16384,
                     data_cnt = 64,
                     data_loop,
                     blk_loop,
           num_bytes = data_size;
        float   show_bytes = 0;
        static char state[2] = {13, 10};

           /* Allocate memory for the data array */
        gen_data = (unsigned char*) malloc (data_size);

           /* Execute the module’s setup commands */
        cmd_exe(set_commands, sizeof(set_commands) / sizeof(char*));
```

Continued on Next Page

```
        /* Call function to check for errors */
    check_error("gen_data (set_commands)");

        /* Calculate 16384 bytes for a single data block */
    for (blk_loop=0; blk_loop<data_size; blk_loop++)
    {
            /* Generate random data */
        gen_data[blk_loop]=(char)(rand());
    }

        /* Disable EOI and EOL to send continuous data to the module;
           then send the data transfer command */
    IOEOI (ISC, 0); IOEOL (ISC, " ", 0);
    IOOUTPUTS (ADDR, out_command, strlen(out_command));

        /* Loop to output data */
    for(data_loop=0; data_loop < data_cnt; data_loop++)
    {
            /* Send 16384 bytes in 64 blocks to module */
        IOOUTPUTB(ADDR, gen_data, num_bytes, 1);

            /* Print total number of bytes transferred */
        printf("Transferred %7.0f bytes\n", (show_bytes += (float)(num_bytes)));
    }

        /* Re-enable EOI and EOL; send CR and LF for end of data */
    IOEOI (ISC, 1); IOEOL (ISC, state, 2);
    IOOUTPUTS(ADDR,"",0);

        /* Free the allocated memory */
    free (gen_data);

        /* Call function to check for errors */
    check_error("gen_data (out_command)");
}
/*****************************************************************************/
void read_data(void)
    /* Function that reads data from the module */
{
    unsigned char   *get_data;

    int     elements = 8192;
    float   get_cnt;
```

Continued on Next Page

```
    char    *set_form = "FORM:DATA PACK",                /* Sets the returned data format */
       *read_cnt = "TRAC:DATA:COUN?  norm_data",       /* Reads data amount */
       *read_cmd = "TRAC:DATA:PART?  norm_data,1,8192"; /* Reads 8192 bytes */

       /* Set the output data format to packed */
    IOOUTPUTS(ADDR, set_form, strlen(set_form));

       /* Read and print amount of data stored in memory module */
    IOOUTPUTS(ADDR, read_cnt, strlen(read_cnt));
    IOENTER(ADDR, &get_cnt);

    printf("\nAmount of data stored in module is: %7.0f bytes\n", get_cnt);

       /* Allocate memory for the data array */
    get_data = (unsigned char*) malloc (elements);

       /* Read data from memory module and store in get_data array */
    IOOUTPUTS (ADDR, read_cmd, strlen(read_cmd));
    IOENTERAB(ADDR, get_data, &elements, 1);

       /* Read LF from last data output to stop data query */
    IOENTER(ADDR, &get_cnt);

       /* Call function to check for errors */
    check_error("read_data");
}
```

Comments

Purpose of the Trace Subsystem

The TRACE | DATA subsystem allocates memory in the Memory Module to allow storage of data.

Retrieving Data After a Power Down

If memory backup is enabled (see Chapter 2), the data will remain in memory after removing power from the module. However, the trace name and memory size will no longer be available. Thus, to read the data in memory, you must redefine the trace name and memory size, and then the amount of memory in bytes you wish to read. For example, to read 1 Mbytes previously stored in memory, use the following commands:

```
TRACE:DEFine  temp, 1048576          Define the trace
TRACE:DATA:COUNt  temp, 1048576      Set the size of data bytes in memory
```

(Note: "temp" is the trace name. It can be different from the trace name defined before power down.)

Order for Stored or Returned Data

The data is stored in the same order received and returned in the same order stored (i.e., received).

Returning All Data that is in Memory

The example program only retrieves some of the data stored in memory using the TRACE:DATA:PARTial command. Use this command for computers or controllers that have insufficient memory to store all the data in the Memory Module. If you wish to read all data bytes from memory, use this command:

```
TRACE:DATA:DATA?
```

Returned Data Formats

The Memory Module can return data using 2 different formats, ASCii and PACKed. The following explains the two formats.

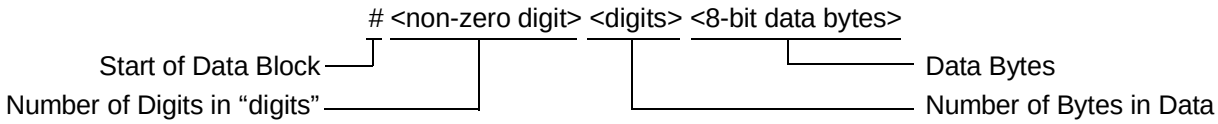
ASCii - This format is the default value and also the slower way to return the stored data. The data is returned as a comma separated list using the NR1 Numeric Response in the form of ±<digit> (e.g., +1,+2,+3,etc.). (NR1 is defined by IEEE-488.2.) Each number represents the value of a data byte in memory. To select the ASCii format, execute:

```
FORMat[:DATA]  ASCii
```

PACKed - This format is the faster way to return the data bytes. The data bytes are returned as Definite Length Arbitrary Block Data, as defined by IEEE-488.2. To select this format, execute:

```
FORMat[:DATA]  PACKed
```

A typical block data format consists of:



where:

- "#" indicates that the data is output in arbitrary block data format
- "<non-zero digit>" is a single digit number that shows the number of digits contained in <digits>; for example, if the <digits> value equals 100 or 4096, the <non-zero digit> value equals 3 and 4, respectively
- "<digits>" shows the number of data bytes to be returned; for example, if 4096 data bytes are returned, <digits> equals 4096
- "<8-bit data bytes>" is the data returned by the module
- A typical example of a data block consisting of 8192 bytes is:

  #48192<data bytes>

To select the PACKed format, execute:

```
FORMat[:DATA]  PACKed
```

Allocating Memory

The Memory Module allocates memory (TRACE | DATA:DEFine command) in multiples of 4 bytes. Thus, if a value is specified that is not a multiple of 4, the module automatically rounds the memory size up to the next value that is at a multiple of 4.

The minimum allocated byte size must be at 4 bytes for any data that is to be output to the Local Bus or to a command module/controller. This is also true for any data received and stored from a command module/controller. However, the exception is that for Local Bus receive data transfers, the minimum size is 16.

Transferring Data Using The Local Bus

The following programs show how the Agilent E1488A Memory Module receives and stores data from the Local Bus, and how to output data to the Local Bus.

Note Whenever an operation on the Local Bus completes, it is a good practice to reset the Local Bus circuitry (VINStrument[:CONFigure]:LBUS:RESet) of the modules that took part of the operation. This ensures proper startup and operation of the Local Bus. (See "Resetting Order of the Local Bus Circuitry of Modules" on page 48 to determine the order in which to reset multiple modules on the Local Bus).

Note The module that is to receive data from the Local Bus *must* always be in the slot immediate to the right of the module that is to output the data, except when passing data through a contiguous Local Bus module (see "Passing Data Through Memory Modules" on page 49).

Storing Data to the Local Bus

The following program shows how the Memory Module receives and stores Local Bus data into memory. The program uses the VINStrument[:CONFigure]:LBUS[:MODE] CONSume command to store the data.

1. **Setup User Module to Output Local Bus Data (DO NOT Initiate the Module to Output Local Bus at this time)**

   Since you select which Output Module to use, no program instructions are given for this step. You must determine how to setup the module. Be sure that the module outputs no data onto the Local Bus at this time.

2. **Disable the Register-level VME Bus Data Transfer Mode**
   VINStrument[:CONFigure]:VME[:MODE] OFF

   This command disables the register-level data transfer on VMEbus. (No need to send this command if the module was previously reset.)

3. **Delete the Trace Currently in Memory**
   TRACE | DATA:DELete:ALL

   This command deletes the trace currently in memory (see next step) to allow for new data to be stored.

4. **Select the Trace Name and Memory Size for Data Storage**
   TRACE | DATE:DEFine *<name>,<length>*

   This command selects the trace name under which the data is to be stored and also sets the memory size (i.e., actual number of data bytes to be stored).

5. **Reset the Local Bus**
   VINStrument[:CONFigure]:LBUS:RESet

   This command resets the Local Bus Controller circuitry.

6. **Set the Local Bus Mode to CONSume**
   VINStrument[:CONFigure]:LBUS[:MODE] CONSume

   This command sets the Local Bus Mode to CONSume to store data bytes into memory.

7. **Enable the *OPC? Query**
   STATus:OPC:INITiate OFF

   This commands allows execution of the *OPC? command.

8. **Initiate Memory Module to Receive Local Bus Data**
   INITiate[:IMMediate]

   This command initiates the Memory Module to receive data from the Local Bus.

9. **Wait for INITiate[:IMMediate] to Complete**
   *OPC?

   This commands halts operation until INITiate[:IMMediate] has completed and the module is ready to receive data. The module returns 1 when ready.

10. **Initiate the Output Module to Output Data**

    Since you select the Output Module to use, no program instructions are given for this step. You must determine how to initiate your module to send data to the Local Bus.

C Program Example (lb_str.c)

The following program configures the module to receive 2097152 bytes of data from a module on the Local Bus.

```
/* lb_str.c -    This program:                                              */
/*               1.) Calls function to setup user module to output data (user */
/*                   writes and supplies the function)                      */
/*               2.) Sends setup commands to store 2097152 bytes into memory */
/*                   from the Local Bus                                     */
/*               3.) Sends commands to initiate the Memory Module to read data */
/*                   from the Local Bus                                     */
/*               4.) Calls function to initiate user module to output the data */
/*                   to the Local Bus (user writes and supplies the function) */
/*                                                                          */
/*  The program assumes a Logical Address of 16 (GPIB Secondary Address of  */
/*  02) for the Memory Module                                               */

      /* Include the following header files */
#include <stdio.h>
#include <string.h>
#include <cfunc.h>        /* This file is from the GPIB Command Library */

#define ADDR 70902L   /* Assign an I/O path between the computer and Memory Module */

      /* Functions */
void store_data(void);
void send_data(void);
void cmd_exe(char *commands[], int length);
void rst_clr(void);
void check_error(char *func_tion);

/*****************************************************************************/
void main(void)
      /* Main function that runs the program */
{
      rst_clr();        /* Reset the Memory Module */
      store_data();  /* Store Local Bus Data into memory */
}

/*****************************************************************************/
void store_data(void)
      /* This function stores data from the Local Bus into memory */
{
```

Continued on Next Page

```
        static char *set_commands[] = /* Use “set_commands” to setup the Memory Module */
        {
        “VINS:CONF:VME:MODE OFF”,       /* Turns register-level data transfer off */

        “TRAC:DEL:ALL”,                 /* Deletes trace from memory */
        “TRAC:DEF lb_data, 2097152",    /* Defines trace name and size */

        “VINS:CONF:LBUS:RES”,           /* Resets Local Bus circuitry */
        “VINS:CONF:LBUS:MODE CONS”,  /* Sets Local Bus Mode to CONSume */

        “STAT:OPC:INIT OFF”             /* Execute *OPC? after INIT is parsed */
        },

        *init_commands[] =          /* Use “init_commands” to start storing data */
        {
        “INIT:IMM”,                     /* Initiate the Memory Module */
        “*OPC?”                         /* Wait for INIT to complete before continuing */
        };

        float      rdy;

           /* Setup user module to output Local Bus data, but do not
              initiate the module to output the data */
/* USER DEFINED FUNCTION */

           /* Execute the module’s setup commands */
        cmd_exe(set_commands, sizeof(set_commands) / sizeof(char*));

           /* Call function to check for errors */
        check_error(“store_data (set_commands)”);

           /* Execute the module’s INIT and *OPC? commands */
        cmd_exe(init_commands, sizeof(init_commands) / sizeof(char*));

           /* Enter *OPC? response; continue when module is ready */
        IOENTER(ADDR, &rdy);

           /* Initiate user module to output data to the Local Bus;
              the module must be in the slot immediately to the left of
              the Memory Module */
/* USER DEFINED FUNCTION */

           /* Call function to check for errors */
        check_error(“store_data (data stored)”);
}
```

Outputting Data to the Local Bus

The following program shows how the Memory Module outputs data from memory to the Local Bus. The program uses the VINStrument[:CONFigure]:LBUS[:MODE] GENerate command to output the data.

The program assumes that data was previously stored into the Memory Module (use the previous lb_str.c program to store the data) and has a current TRACe name and length (TRACe:DEFine <*name*>,<*length*>).

1. **Disable the Register-level VME Bus Data Transfer Mode**
   VINStrument[:CONFigure]:VME[:MODE] OFF

   This command disables the register-level data transfer on VMEbus. (No need to send this command if the mode was previously disabled.)

2. **Reset the Local Bus**
   VINStrument[:CONFigure]:LBUS:RESet

   This command resets the Local Bus Controller circuitry.

3. **Set the Local Bus Mode to GENerate**
   VINStrument[:CONFigure]:LBUS[:MODE] GENerate

   This command sets the Local Bus Mode to GENerate to output data bytes from memory.

4. **Enable the *OPC? Query**
   STATus:OPC:INITiate OFF

   This commands allows execution of the *OPC? command.

5. **Setup Input Module to Receive Data**

   Since you select which Input Module to use, no program instructions are given for this step. You must determine how to setup your module to receive data from the Local Bus.

6. **Initiate Input Module to Receive Data**

   Since you select which Input Module to use, no program instructions are given for this step. You must determine how to initiate your input module to receive data from the Local Bus.

7. **Initiate Memory Module to Output Local Bus Data**
   INITiate[:IMMediate]

   This command initiates the Memory Module to output data to the Local Bus.

8. **Wait for INITiate[:IMMediate] to Complete**
   *OPC?

   This commands halts operation until INITiate[:IMMediate] has completed and the module then sends the data.

C Program Example (lb_out.c)

The following program configures the module to output 2097152 bytes of data from memory to a module on the Local Bus.

```
/* lb_out.c -   This program:                                              */
/*              1.) Sends setup commands to output 2097152 bytes from memory */
/*                  to the Local Bus                                       */
/*              2.) Calls function to setup user module to read data from the */
/*                  Memory Module (user writes and supplies the function)  */
/*              3.) Calls function to initiate user module to read the data */
/*                  from the Local Bus (user writes and supplies the function) */
/*              4.) Sends commands to initiate the Memory Module to output the */
/*                  data to the Local Bus                                  */
/*                                                                         */
/*  The program assumes a Logical Address of 16 (GPIB Secondary Address of */
/*  02) for the Memory Module                                              */

      /* Include the following header files */
#include <stdio.h>
#include <string.h>
#include <cfunc.h>        /* This file is from the GPIB Command Library */

#define ADDR 70902L   /* Assign an I/O path between the computer and Memory Module */

      /* Functions */
void store_data(void);
void send_data(void);
void cmd_exe(char *commands[], int length);
void rst_clr(void);
void check_error(char *func_tion);

/*************************************************************************/
void main(void)
      /* Main function that runs the program */
{
      rst_clr();        /* Reset the Memory Module */
      send_data();  /* Outputs stored data from memory */
}
```

Continued on Next Page

```
/*****************************************************************************/
void send_data(void)
        /* This functions outputs data from memory to the Local Bus */
{

        static char *send_commands[] = /* Use “send_commands” to setup the Memory Module*/
        {
        “VINS:CONF:VME:MODE OFF”,       /* Turns register-level data transfer off */

        “VINS:CONF:LBUS:RES”,           /* Resets Local Bus circuitry */
        “VINS:CONF:LBUS:MODE GEN”,      /* Turns Local Bus Mode to GENerate */

        “STAT:OPC:INIT OFF”             /* Execute *OPC? after INIT is parsed */
        },

        *init_commands[] =          /* Use “init_commands” to start outputting data */
        {
        “INIT:IMM”,                     /* Initiate the Memory Module */
        “*OPC?”                         /* Wait for INIT to complete before continuing */
        };

        float      rdy;

           /* Execute the module’s setup commands */
        cmd_exe(send_commands, sizeof(send_commands) / sizeof(char*));

           /* Call function to check for errors */
        check_error(“send_data (send_commands)”);

           /* Setup user module to receive Local Bus data, but do not
              enable the module to receive the data */
/* USER DEFINED FUNCTION */

           /* Initiate user module to receive data from the Local Bus */
/* USER DEFINED FUNCTION */

           /* Initiate module to output data from memory onto the Local Bus */
        cmd_exe(init_commands, sizeof(init_commands) / sizeof(char*));

           /* Enter *OPC? response; continue when module is ready */
        IOENTER(ADDR, &rdy);

           /* Call function to check for errors */
        check_error(“send_data (init_commands)”);
}
```

Comments

How Data is Transferring on the Local Bus

The Local Bus in an VXIbus system is located on the P2 connector. Thus, the module that receives data must be in the slot immedeately to the right of the module that outputs the data (see Chapter 4 for more information on the Local Bus). Figure 3-1 shows a typical example on how data is transferred over the Local Bus.

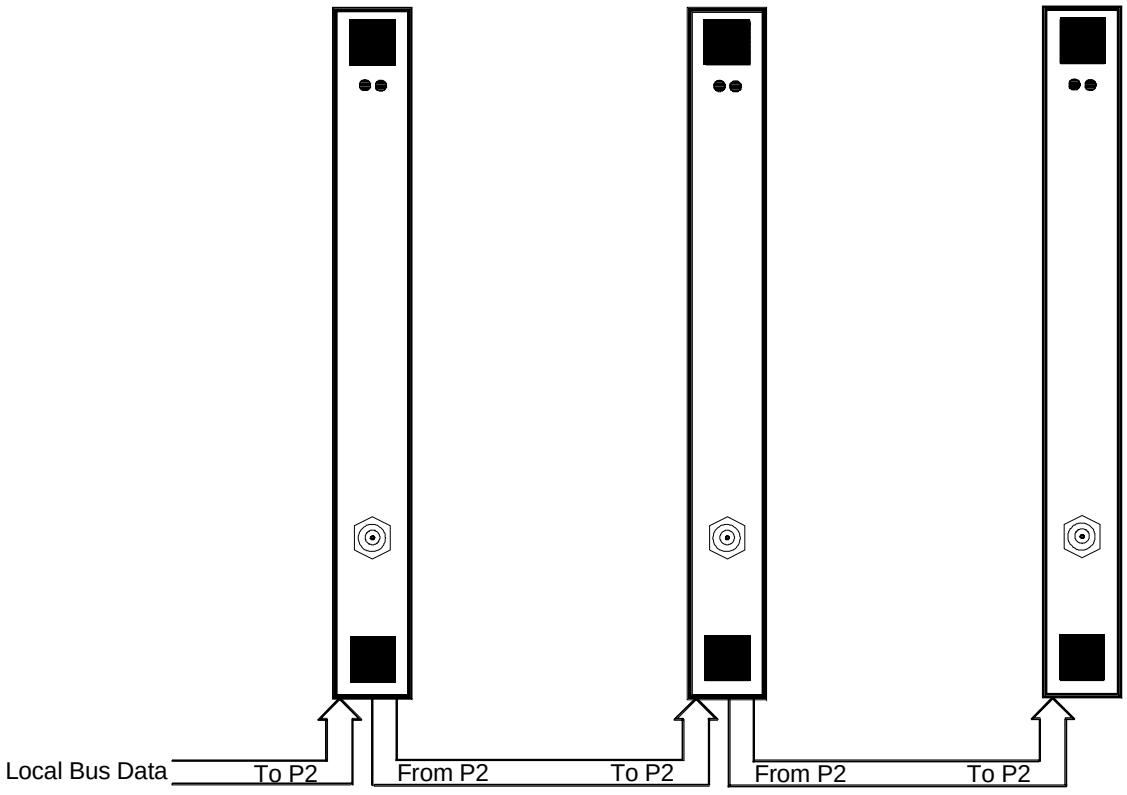


Figure 3-1. Typical Local Bus Operation

Aborting Data Transfer Prematurely

The Memory Module continues storing data until it reaches the allocated memory size set by the TRACE | DATA:DEFine command. If the data bytes on the Local Bus are less than the allocated size, the module will wait until it receives the correct amount of bytes. If the output device or module is unable to output the correct amount of data, send the ABORt (or *RST) command to stop the Memory Module from waiting for more data.

Resetting Order of the Local Bus Circuitry of Modules

When using multiple modules on the Local Bus, reset the Local Bus circuitry starting with the leftmost module first and ending with the rightmost module last.

Using Multiple Memory Modules on the Local Bus

The Memory Module has the following Local Bus capabilities when using multiple modules:

- pass data through one or more modules and store it in another module
- store data in several modules
- output data from several modules

Passing Data Through Memory Modules

The following program shows how to pass Local Bus data from a Memory Module through another Memory Module, and then output the data to the Local Bus. The first module outputs the data and the second module passes the data to the Local Bus. The program uses the VINStrument[:CONFigure]:LBUS[:MODE] command with the GENerate and PIPEline parameters to output and pass the data, respectively.

The program assumes that data was previously stored into the Memory Module (use the previous lb_str.c program to store the data) and has a current TRACe name and length (TRACe:DEFine <*name*>,<*length*>).

Note For simplicity purposes, the first Memory Module in the following program outputs the data to the Local Bus to the second Memory Module. However, any module that can output data to the Local Bus can be used in place of the first Memory Module. If using a different module, substitutethe the appropriate setup commands of the replacement module for the setup commands of the first Memory Module.

---

1. **Module 1: Disable the Register-level VME Bus Data Transfer Mode**
   VINStrument[:CONFigure]:VME[:MODE] OFF

   This command disables the register-level data transfer on VMEbus. (No need to send this command if the module was previously reset.)

2. **Module 1: Reset the Local Bus**
   VINStrument[:CONFigure]:LBUS:RESet

   This command resets the Local Bus Controller circuitry.

3. **Set the Local Bus Mode to GENerate**
   VINStrument[:CONFigure]:LBUS[:MODE] GENerate

   This command sets the Local Bus Mode to GENerate to output data bytes from memory.

4. **Module 1: Enable the *OPC? Query**
   STATus:OPC:INITiate OFF

   This commands allows execution of the *OPC? command.

5. **Module 2: Disable the Register-level VME Bus Data Transfer Mode**
   VINStrument[:CONFigure]:VME[:MODE] OFF

   This command disables the register-level data transfer on VMEbus. (No need to send this command if the module was previously reset.)

6. **Module 2: Reset the Local Bus**
   VINStrument[:CONFigure]:LBUS:RESet

   This command resets the Local Bus Controller circuitry.

7. **Module 2: Set the Local Bus Mode to PIPeline**
   VINStrument[:CONFigure]:LBUS[:MODE] PIPeline

   This command sets the Local Bus Mode to PIPeline to pass the data from the Local Bus, through the module, and to the Local Bus.

8. **Module 2: Enable the *OPC? Query**
   STATus:OPC:INITiate OFF

   This commands allows execution of the *OPC? command.

9. **Setup Input Module to Receive Data**

   Since you select which Input Module to use, no program instructions are given for this step. You must determine how to setup your module to receive data from the Local Bus.

10. **Initiate Input Module to Receive Data**

    Since you select which Input Module to use, no program instructions are given for this step. You must determine how to initiate your input module to receive data from the Local Bus.

11. **Module 2: Initiate Memory Module to Pass Local Bus Data**
    INITiate[:IMMediate]

    This command initiates the Memory Module to pass Local Bus data through the module.

12. **Module 2: Wait for INITiate[:IMMediate] to Complete**
    *OPC?

    This commands halts operation until INITiate[:IMMediate] has completed and the module is ready to receive data. The module returns 1 when ready.

13. **Module 1: Initiate Memory Module to Output Local Bus Data**
    INITiate[:IMMediate]

    This command initiates the Memory Module to output data to the Local Bus.

14. **Module 1: Wait for INITiate[:IMMediate] to Complete**
    *OPC?

    This commands halts operation until INITiate[:IMMediate] has completed and the module is ready to output data. The module returns 1 when ready.

C Program Example (lb_pipe.c)

This program passes data bytes from a Memory Module to the Local Bus, through another memory Mmodule, and then to yhe Local Bus.

```
/* lb_pipe.c -  This program                                                    */
/*              1.) Sends setup commands to the first Memory Module to output   */
/*                  2097152 bytes of data to the Local Bus                      */
/*              2.) Sends setup commands to the second Memory Module to pass    */
/*                  the data from the Local Bus through it                      */
/*              3.) Calls function to setup user module to receive the data     */
/*                  from the Local Bus (user writes and supplies the function)  */
/*              4.) Calls function to initiate user module to receive the data  */
/*                  from the Local Bus (user writes and supplies the function)  */
/*              5.) Sends commands to initiate the second Memory Module to pass */
/*                  data from the Local Bus through it                          */
/*              6.) Sends commands to initiate the first Memory Module to       */
/*                  output data to the Local Bus to the second Memory Module    */
/*                                                                              */
/* The program assumes the following Logical Addresses (Secondary Addresses):   */
/*      Module 1: 16 (02)                                                       */
/*      Module 2: 24 (03)                                                       */

      /* Include the following header files */
#include <stdio.h>
#include <string.h>
#include <cfunc.h>        /* This file is from the GPIB Command Library */
```

Continued on Next Page

```
#define ADDR1 70902L  /* Assign an I/O path between the computer and Memory Module 1 */
#define ADDR2 70903L  /* Assign an I/O path between the computer and Memory Module 2 */

        /* Functions */
void send_data(void);
void cmd_exe(long addr, char *commands[], int length);
void rst_clr(long addr);
void check_error(long addr, char *func_tion);

/*****************************************************************************/
void main(void)
        /* Main function that runs the program */
{
        rst_clr(ADDR1);      /* Reset Memory Module 1 */
        rst_clr(ADDR2);      /* Reset Memory Module 2 */
        send_data();         /* Send and pass data */
}
/*****************************************************************************/
        /* Function that passes and stores data */
void send_data(void)
{
        static char *set1_commands[] = /* Use "set1_commands" to setup module 1 */
        {
        "VINS:CONF:VME:MODE OFF",      /* Turns register-level data transfer off */

        "VINS:CONF:LBUS:RES",          /* Resets Local Bus circuitry */
        "VINS:CONF:LBUS:MODE GEN",     /* Sets Local Bus Mode to GENerate */

        "STAT:OPC:INIT OFF"            /* Execute *OPC? after INIT is parsed */
        },

        *set2_commands[] =          /* Use "set2_commands" to setup module 2 */
        {
        "VINS:CONF:VME:MODE OFF",      /* Turns register-level data transfer off */

        "VINS:CONF:LBUS:RES",          /* Resets Local Bus circuitry */
        "VINS:CONF:LBUS:MODE PIP",     /* Sets Local Bus Mode to PIPeline */

        "STAT:OPC:INIT OFF"            /* Execute *OPC? after INIT is parsed */
        },

        *init_commands[] =          /* Use "init_commands" to start passing/storing data */
        {
        "INIT:IMM",                    /* Initiate the Memory Module */
        "*OPC?"                        /* Wait for INIT to complete before continuing */
        };
```

Continued on Next Page

```
        float      rdy;

            /* Execute module 1 setup commands */
        cmd_exe(ADDR1, set1_commands, sizeof(set1_commands) / sizeof(char*));

            /* Call function to check for errors */
        check_error(ADDR1, “send_data (set1_commands)”);

            /* Execute module 2 setup commands */
        cmd_exe(ADDR2, set2_commands, sizeof(set2_commands) / sizeof(char*));

            /* Call function to check for errors */
        check_error(ADDR2, “send_data (set2_commands)”);

            /* Setup user module to receive Local Bus data */
/* USER DEFINED FUNCTION */

            /* Initiate user module to receive data from the Local Bus */
/* USER DEFINED FUNCTION */

            /* Execute module 2 INIT and *OPC? commands */
        cmd_exe(ADDR2, init_commands, sizeof(init_commands) / sizeof(char*));

            /* Enter *OPC? response; continue when module is ready */
        IOENTER(ADDR2, &rdy);

            /* Execute module 1 INIT and *OPC? commands */
        cmd_exe(ADDR1, init_commands, sizeof(init_commands) / sizeof(char*));

            /* Enter *OPC? response; continue when module is ready */
        IOENTER(ADDR1, &rdy);
}
```

Storing Data into Multiple Modules

For applications with very large data storage, use multiple Memory Modules together to store the data. The following program shows how to store data into two Memory Modules. The program uses the VINStrument[:CONFigure]:LBUS[:MODE] command with the STRip and CONSume parameters to store the data in module 1 and module 2, respectively.

The Memory Module that uses the STRip parameter stores the first n bytes of data and then passes through the remainder (n equals the amount of allocated memory in bytes).

1. **Setup User Module to Output Local Bus Data (DO NOT Initiate the Module to Output Local Bus at this time)**

   Since you select which Output Module to use, no program instructions are given for this step. You must determine how to setup the module. Be sure that the module outputs no data onto the Local Bus (see Chapter 4 for information on Local Bus operation).

2. **Module 1: Disable the Register-level VME Bus Data Transfer Mode**
   VINStrument[:CONFigure]:VME[:MODE] OFF

   This command disables the register-level data transfer on VMEbus. (No need to send this command if the module was previously reset.)

3. **Module 1: Delete the Trace Currently in Memory**
   TRACE | DATA:DELete:ALL

   This command deletes the trace currently in memory (see next step) to allow for new data to be stored. (Note that the actual data bytes are not deleted from memory, only the trace is deleted.)

4. **Module 1: Select the Trace Name and Memory Size for Data Storage**
   TRACE | DATE:DEFine *<name>*,*<length>*

   This command selects the trace name under which the data is to be stored and also sets the memory size (i.e., maximum number of data bytes to be stored).

5. **Module 1: Reset the Local Bus**
   VINStrument[:CONFigure]:LBUS:RESet

   This command resets the Local Bus Controller circuitry.

6. **Module 1: Set the Local Bus Mode to STRip**
   VINStrument[:CONFigure]:LBUS[:MODE] STRip

   This command sets the Local Bus Mode to STRip to store data bytes into memory, and when memory is full, pass the rest of the data bytes to the next Memory Module.

7. **Module 1: Enable the *OPC? Query**
   STATus:OPC:INITiate OFF

   This commands allows execution of the *OPC? command.

8. **Module 2: Disable the Register-level VME Bus Data Transfer Mode**
   VINStrument[:CONFigure]:VME[:MODE] OFF

   This command disables the register-level data transfer on VMEbus. (No need to send this command if the module was previously reset.)

9. **Module 2: Delete the Trace Currently in Memory**
   TRACE | DATA:DELete:ALL

   This command deletes the trace currently in memory (see next step) to allow for new data to be stored.

10. **Module 2: Select the Trace Name and Memory Size for Data Storage**
    TRACE | DATE:DEFine *<name>,<length>*

    This command selects the trace name under which the data is to be stored and also sets the memory size (i.e., maximum number of data bytes to be stored).

11. **Module 2: Reset the Local Bus**
    VINStrument[:CONFigure]:LBUS:RESet

    This command resets the Local Bus Controller circuitry.

12. **Module 2: Set the Local Bus Mode to CONSume**
    VINStrument[:CONFigure]:LBUS[:MODE] CONSume

    This command sets the Local Bus Mode to CONSume to store data bytes into memory.

13. **Module 2: Enable the *OPC? Query**
    STATus:OPC:INITiate OFF

    This commands allows execution of the *OPC? command.

14. **Module 2: Initiate Memory Module to Receive Local Bus Data**
    INITiate[:IMMediate]

    This command initiates the Memory Module to receive data from the Local Bus.

15. **Module 2: Wait for INITiate[:IMMediate] to Complete**
    *OPC?

    This commands halts operation until INITiate[:IMMediate] has completed and the module is ready to receive data. The module returns 1 when ready.

16. **Module 1: Initiate Memory Module to Receive Local Bus Data**
    INITiate[:IMMediate]

    This command initiates the Memory Module to receive data from the Local Bus.

17. **Module 1: Wait for INITiate[:IMMediate] to Complete**
    *OPC?

    This commands halts operation until INITiate[:IMMediate] has completed and the module is ready to pass data. The module returns 1 when ready.

18. **Initiate the Output Module to Output Data**

    Since you select the Output Module to use, no program instructions are given for this step. You must determine how to initiate your module to send data to the Local Bus.

C Program Example (lb_mstr.c)

This program stores data bytes from the Local Bus into two modules.

```
/* lb_mstr.c -  This program                                                    */
/*              1.) Calls function to setup user module to output data (user    */
/*                  writes and supplies the function)                           */
/*              2.) Sends setup commands to the first Memory Module to store    */
/*                  the first 1       048576 bytes into memory from the Local Bus        */
/*              3.) Sends setup commands to the second Memory Module store      */
/*                  the second 1048576 bytes into memory from the Local Bus     */
/*              4.) Sends commands to initiate the second Memory Module to read */
/*                  data from the Local Bus                                     */
/*              5.) Sends commands to initiate the first Memory Module to read  */
/*                  data from the Local Bus                                     */
/*              6.) Calls function to initiate user module to output the data   */
/*                  to the Local Bus (user writes and supplies the function)    */
/*                                                                              */
/* The program assumes the following Logical Addresses (Secondary Addresses): */
/*      Module 1: 16 (02)                                                       */
/*      Module 2: 24 (03)                                                       */

        /* Include the following header files */
#include <stdio.h>
#include <string.h>
#include <cfunc.h>          /* This file is from the GPIB Command Library */

#define ADDR1 70902L  /* Assign an I/O path between the computer and Memory Module 1 */
#define ADDR2 70903L  /* Assign an I/O path between the computer and Memory Module 2 */

        /* Functions */
void store_data(void);
void cmd_exe(long addr, char *commands[], int length);
void rst_clr(long addr);
void check_error(long addr, char *func_tion);

/*******************************************************************************/
void main(void)
        /* Main function that runs the program */
{
        rst_clr(ADDR1);      /* Reset Memory Module 1 */
        rst_clr(ADDR2);      /* Reset Memory Module 2 */
        store_data();        /* Store data in all modules */
}
```

Continued on Next Page

```
/*****************************************************************************/
void store_data(void)
      /* Function that stores the data into both modules */
{
      static char *set1_commands[] = /* Use “set1_commands” to setup module 1 */
      {
      “VINS:CONF:VME:MODE OFF”,      /* Turns register-level data transfer off */

      “TRAC:DEL:ALL”,                /* Deletes trace from memory */
      “TRAC:DEF lb_data, 1048576",   /* Defines trace name and size */

      “VINS:CONF:LBUS:RES”,          /* Resets Local Bus circuitry */
      “VINS:CONF:LBUS:MODE STR”,     /* Sets Local Bus Mode to STRip */

      “STAT:OPC:INIT OFF”            /* Execute *OPC? after INIT is parsed */
      },

      *set2_commands[] =             /* Use “set2_commands” to setup module 2 */
      {
      “VINS:CONF:VME:MODE OFF”,      /* Turns register-level data transfer off */

      “TRAC:DEL:ALL”,                /* Deletes trace from memory */
      “TRAC:DEF lb_data, 1048576",   /* Defines trace name and size */

      “VINS:CONF:LBUS:RES”,          /* Resets Local Bus circuitry */
      “VINS:CONF:LBUS:MODE CONS”,    /* Sets Local Bus Mode to CONSume */

      “STAT:OPC:INIT OFF”            /* Execute *OPC? after INIT is parsed */
      },

      *init_commands[] =             /* Use “init_commands” to start passing/storing data */
      {
      “INIT:IMM”,                    /* Initiate the Memory Module */
      “*OPC?”                        /* Wait for INIT to complete before continuing */
      };

      float     rdy;

         /* Setup user module to output Local Bus data, but do not
            initiate the module to output the data */
/* USER DEFINED FUNCTION */

         /* Execute module 1 setup commands */
      cmd_exe(ADDR1, set1_commands, sizeof(set1_commands) / sizeof(char*));
```

Continued on Next Page

```
        /* Call function to check for errors */
    check_error(ADDR1, “store_data (set1_commands)”);

        /* Execute module 2 setup commands */
    cmd_exe(ADDR2, set2_commands, sizeof(set2_commands) / sizeof(char*));

        /* Call function to check for errors */
    check_error(ADDR2, “store_data (set2_commands)”);

        /* Execute module 2 INIT and *OPC? commands */
    cmd_exe(ADDR2, init_commands, sizeof(init_commands) / sizeof(char*));

        /* Enter *OPC? response; continue when module is ready */
    IOENTER(ADDR2, &rdy);

        /* Execute module 1 INIT and *OPC? commands */
    cmd_exe(ADDR1, init_commands, sizeof(init_commands) / sizeof(char*));

        /* Enter *OPC? response; continue when module is ready */
    IOENTER(ADDR1, &rdy);

        /* Initiate user module to output data to the Local Bus */
/* USER DEFINED FUNCTION */

        /* Call function to check for errors */
    check_error(ADDR1, “store_data (MODULE 1)”);

        /* Call function to check for errors */
    check_error(ADDR2, “store_data (MODULE 2)”);
}
```

Outputting Data from Multiple Modules

The following program shows how to output data from two Memory Modules. The program uses the VINStrument[:CONFigure]:LBUS[:MODE] command with the GENerate and APPend parameters to output the data from module 1 and module 2, respectively.

The program assumes that data was previously stored into the Memory Modules and that they have a current TRACe names and lengths (TRACe:DEFine <*name*>,<*length*>).

1. **Module 1: Disable the Register-level VME Bus Data Transfer Mode**
   VINStrument[:CONFigure]:VME[:MODE] OFF

   This command disables the register-level data transfer on VMEbus. (No need to send this command if the mode was previously disabled.)

2. **Module 1: Reset the Local Bus**
   VINStrument[:CONFigure]:LBUS:RESet

   This command resets the Local Bus Controller circuitry.

3. **Module 1: Set the Local Bus Mode to GENerate**
   VINStrument[:CONFigure]:LBUS[:MODE] GENerate

   This command sets the Local Bus Mode to GENerate to output data bytes from memory.

4. **Module 1: Enable the *OPC? Query**
   STATus:OPC:INITiate OFF

   This commands allows execution of the *OPC? command.

5. **Module 2: Disable the Register-level VME Bus Data Transfer Mode**
   VINStrument[:CONFigure]:VME[:MODE] OFF

   This command disables the register-level data transfer on VMEbus. (No need to send this command if the mode was previously disabled.)

6. **Module 2: Reset the Local Bus**
   VINStrument[:CONFigure]:LBUS:RESet

   This command resets the Local Bus Controller circuitry.

7. **Module 2: Set the Local Bus Mode to APPend**
   VINStrument[:CONFigure]:LBUS[:MODE] GENerate

   This command sets the Local Bus Mode to APPEnd to append the data to the data previously output by Module 1.

8. **Module 1: Enable the *OPC? Query**
   STATus:OPC:INITiate OFF

   This commands allows execution of the *OPC? command.

9. **Setup Input Module to Receive Data**

   Since you select which Input Module to use, no program instructions are given for this step. You must determine how to setup your module to receive data from the Local Bus.

10. **Initiate Input Module to Receive Data**

    Since you select which Input Module to use, no program instructions are given for this step. You must determine how to initiate your input module to receive data from the Local Bus.

11. **Module 2: Initiate Memory Module to Output Local Bus Data**
    INITiate[:IMMediate]

    This command initiates the Memory Module to output data to the Local Bus.

12. **Module 2: Wait for INITiate[:IMMediate] to Complete**
    *OPC?

    This commands halts operation until INITiate[:IMMediate] has completed and the module then sends the data.

13. **Module 1: Initiate Memory Module to Output Local Bus Data**
    INITiate[:IMMediate]

    This command initiates the Memory Module to output data to the Local Bus.

14. **Module 1: Wait for INITiate[:IMMediate] to Complete**
    *OPC?

    This commands halts operation until INITiate[:IMMediate] has completed and the module then sends the data.

C Program Example (lb_mout.c)

This program outputs data bytes to the Local Bus from two modules.

```
/* lb_mout.c - This program                                                    */
/*             1.) Sends setup commands to Module 1 to output data bytes from  */
/*                 memory to the Local Bus                                     */
/*             1.) Sends setup commands to Module 2 to output data bytes from  */
/*                 memory to the Local Bus                                     */
/*             3.) Calls function to setup user module to read data from the   */
/*                 Memory Module (user writes and supplies the function)       */
/*             4.) Calls function to initiate user module to read the data     */
/*                 from the Local Bus (user writes and supplies the function)  */
/*             5.) Sends commands to initiate Memory Module 2 to output the    */
/*                 data to the Local Bus                                       */
/*             6.) Sends commands to initiate Memory Module 1 to output the    */
/*                 data to the Local Bus                                       */
/* The program assumes the following Logical Addresses (Secondary Addresses): */
/*     Module 1: 16 (02)                                                       */
/*     Module 2: 24 (03)                                                       */

      /* Include the following header files */
#include <stdio.h>
#include <string.h>
#include <cfunc.h>        /* This file is from the GPIB Command Library */

#define ADDR1 70902L  /* Assign an I/O path between the computer and Memory Module 1 */
#define ADDR2 70903L  /* Assign an I/O path between the computer and Memory Module 2 */

      /* Functions */
void out_data(void);
void cmd_exe(long addr, char *commands[], int length);
void rst_clr(long addr);
void check_error(long addr, char *func_tion);

/*****************************************************************************/
void main(void)
      /* Main function that runs the program */
{
      rst_clr(ADDR1);     /* Reset Memory Module 1 */
      rst_clr(ADDR2);     /* Reset Memory Module 2 */
      out_data();     /* Outputs data from all modules */
}
```

Continued on Next Page

```
/*****************************************************************************/
void out_data(void)
        /* Function that outputs data from all Memory Modules */
{
        static char *set1_commands[] = /* Use “set1_commands” to setup module 1 */
        {
        “VINS:CONF:VME:MODE OFF”,       /* Turns register-level data transfer off */

        “VINS:CONF:LBUS:RES”,           /* Resets Local Bus circuitry */
        “VINS:CONF:LBUS:MODE GEN”,      /* Sets Local Bus Mode to GENerate */

        “STAT:OPC:INIT OFF”             /* Execute *OPC? after INIT is parsed */
        },

        *set2_commands[] =              /* Use “set2_commands” to setup module 2 */
        {
        “VINS:CONF:VME:MODE OFF”,       /* Turns register-level data transfer off */

        “VINS:CONF:LBUS:RES”,           /* Resets Local Bus circuitry */
        “VINS:CONF:LBUS:MODE APP”,      /* Sets Local Bus Mode to APPend */

        “STAT:OPC:INIT OFF”             /* Execute *OPC? after INIT is parsed */
        },

        *init_commands[] =              /* Use “init_commands” to start passing/storing data */
        {
        “INIT:IMM”,                     /* Initiate the Memory Module */
        “*OPC?”                         /* Wait for INIT to complete before continuing */
        };

        float      rdy;

           /* Execute module 1 setup commands */
        cmd_exe(ADDR1, set1_commands, sizeof(set1_commands) / sizeof(char*));

           /* Call function to check for errors */
        check_error(ADDR1, “store_data (set1_commands)”);

           /* Execute module 2 setup commands */
        cmd_exe(ADDR2, set2_commands, sizeof(set2_commands) / sizeof(char*));

           /* Call function to check for errors */
        check_error(ADDR2, “store_data (set2_commands)”);
```

Continued on Next Page

```
        /* Setup user module to receive Local Bus data, but do not
           enable the module to receive the data */
/* USER DEFINED FUNCTION */

        /* Initiate user module to receive data from the Local Bus */
/* USER DEFINED FUNCTION */

        /* Execute module 2 INIT and *OPC? commands */
      cmd_exe(ADDR2, init_commands, sizeof(init_commands) / sizeof(char*));

        /* Enter *OPC? response; continue when module is ready */
      IOENTER(ADDR2, &rdy);

        /* Execute module 1 INIT and *OPC? commands */
      cmd_exe(ADDR1, init_commands, sizeof(init_commands) / sizeof(char*));

        /* Enter *OPC? response; continue when module is ready */
      IOENTER(ADDR1, &rdy);

        /* Call function to check for errors */
      check_error(ADDR1, “store_data (MODULE 1)”);

        /* Call function to check for errors */
      check_error(ADDR2, “store_data (MODULE 2)”);
}
```

Comments

Configuration and Initiation Order for Multiple Modules

Data Output from a Single Module to Multiple Memory Modules

Use the following order to configure and initiate a single user selected module to output data and multiple Memory Modules to receive the data. Configuration of *all* modules includes sending the VINStrument:[:CONFigure]:LBUS:RESet command (or equivalent, dependent on your module) to reset the Local Bus.

The following example uses two memory Modules to receive data and one user selected module to output data (the same order applies if using more than two Memory Modules that are to receive data).

1. Configure the Output Module to output data
2. Configure Module 1 to receive data
3. Configure Module 2 to receive data
4. Initiate Module 2
5. Initiate Module 1
6. Initiate the Output Module

Data Output from Multiple Memory Modules to a Single Module

Use the following order to configure multiple Memory Modules to output data and a single user selected module receive the data. Configuration of *all* modules includes sending the VINStrument:[:CONFigure]:LBUS:RESet command (or equivalent; dependent on your module) to reset the Local Bus.

The following example uses two Memory Modules to output data and one user selected module to receive data (the same order applies if using more than two Memory Modules to output data).

1. Configure Memory Module 1 to output data
2. Configure Memory Module 2 to output data
3. Configure the Input Module to receive data
4. Initiate the Input Module
5. Initiate Memory Module 2
6. Initiate Memory Module 1

Using More than Two Modules to Receive Data

The example program (lb_mstr.c) in this section uses two modules to receive data. If using more than two modules, configure the center modules the same as the leftmost module. Configure the modules as follows:

Module 1: VINStrument[:CONFigure]:LBUS[:MODE} STRip

Module 2: VINStrument[:CONFigure]:LBUS[:MODE} STRip

Module 3: VINStrument[:CONFigure]:LBUS[:MODE} CONSume

Different Ways to Output Data with Multiple Modules

There are different ways for multiple Memory Modules to output data. The following shows the different methods. To understand these methods, you need some knowledge of Local Bus operations (see Chapter 4 for more information on Local Bus operations and definitions).

Outputting Data as Separate Blocks Using Two Modules

The output program example (lb_mout.c) in this section of the chapter uses this method to output data. With this method, each module outputs a block of data with both an end-of-block and an end-of-frame indication. Use these commands:

Module 1: VINStrument[:CONFigure]:LBUS[:MODE} GENerate

The module outputs all data and then sends both an end-of-block and end-of-frame indication.

Module 2: VINStrument[:CONFigure]:LBUS[:MODE} APPend

This module passes data through and, after it receives an end-of-frame indication, it then strips off the end-of-frame and outputs its data. It sends both an end-of-block and end-of-frame indication with the last data byte.

Outputting Data as Separate Blocks Using More than Two Memory Modules

This method is similar to the previous method. Use these commands:

Module 1: VINStrument[:CONFigure]:LBUS[:MODE} GENerate

The module outputs all data sending both an end-of-block and end-of-frame indication with the last data byte.

Module 2: VINStrument[:CONFigure]:LBUS[:MODE} APPend

This module passes data through until it receives an end-of-frame indication. It then strips off the end-of-frame and outputs its data sending both an end-of-block and end-of-frame indication with the last data byte.

Module 3: VINStrument[:CONFigure]:LBUS[:MODE} APPend

This module passes data through until it receives an end-of-frame indication. It then strips off the end-of-frame and outputs its data sending both an end-of-block and end-of-frame indication with the last data byte.

Outputting Data as One Block Using Two Memory Modules

In this method, the first module outputs a block of data with only an end-of-frame, but no end-of-block indication. The last module outputs both an end-of-block and end-of-frame indication. Use these commands:

Module 1: VINStrument[:CONFigure]:LBUS[:MODE} MGENerate

This command outputs all data sending only an end-of-frame indication with the last data byte.

Module 2: VINStrument[:CONFigure]:LBUS[:MODE} APPend

This module passes data through until it receives an end-of-frame indication. It then strips off the end-of-frame and outputs its data sending both an end-of-block and end-of-frame indication with the last data byte.

Outputting Data as One Block Using More than Two Memory Modules

This method is similar to the previoud method. However, the center modules use a different command from the commands listed above. Use these commands:

Module 1: VINStrument[:CONFigure]:LBUS[:MODE} MGENerate

This command outputs all data sending only an end-of-frame indication with the last data byte.

Module 2: VINStrument[:CONFigure]:LBUS[:MODE} MAPPend

This command passes through data until it receives an end-of-frame indication. It then strips off the end-of-frame and outputs its data. It sends an end-of-frame indication, but no end-of-block indication with the last data byte.

Module 3: VINStrument[:CONFigure]:LBUS[:MODE} APPend

This module passes data through until it receives an end-of-frame indication. It then strips off the end-of-frame and outputs its data sending both an end-of-block and end-of-frame indication with the last data byte.

Outputting Blocks of Data in Reverse Using Multiple Memory Modules

Multiple modules usually output data starting from the leftmost module to the rightmost module. However, you can output data starting from the rightmost module to the leftmost module.

For example, if you have three modules, you can have Module 3 (the rightmost module) output data first, then Module 2 (the center module) output data next, and then Module 1 (the leftmost module) output data last. This method outputs individual blocks of data and not in one single block (see "Outputting Data as One Block Using Two Memory Modules" on the previous page). Use these commands to configure the modules:

Module 1: VINStrument[:CONFigure]:LBUS[:MODE} GENerate

The module outputs all data and then sends both an end-of-block and end-of-frame indication.

Module 2: VINStrument[:CONFigure]:LBUS[:MODE} INSert

This module outputs data and, after all data is output, it sends an end-of-block but no end-of-frame indication. The module then passes data through until it receives an end-of-frame indication.

Module 3: VINStrument[:CONFigure]:LBUS[:MODE} INSert

This module outputs data and, after all data is output, it sends an end-of-block but no end-of-frame indication. The module then passes data through until it receives an end-of-frame indication.

Use this order to initiate the modules:

1. Inititate Memory Module 1
2. Inititate Memory Module 2
3. Inititate Memory Module 3

Chapter 4
Understanding the Memory Module

Chapter Contents

This chapter shows you how the Agilent E1488A Instrument Memory Module (Memory Module) operates on the Local Bus (LB) and how the module's SCPI Status System operates. The main sections of this chapter are:

- Local Bus Data Transfers. 69
  - Local Bus Description 69
  - How to Transfer Data 70
  - Local Bus Operating Modes. 71
- The SCPI Status System 73
  - Status System Overview. 73
  - The Standard Event Group 74
  - The Operation Status Group. 75
  - The Questionable Data Group 77
  - The Status Byte Group 77

Local Bus Data Transfers

This section describes the use of the VXIbus backplane Local Bus (LB). The Agilent E1488A Instrument Memory Module (Memory Module) uses the Local Bus for high-speed (80 Mbytes/second) data transfers between two or more modules installed in adjacent mainframe slots.

Local Bus Description

The following describes the Local Bus operation for the Memory Module.

Local Bus Data/Control Lines

The Local Bus is a daisy-chained bus that consists of a set of 12 signal lines on the backplane P2 connector. The 12 signal lines consist of the following (Figure 4-1 illustrates the lines) :

- 8 data lines (LDATA0 - LDATA7)
- 2 handshake lines (LDAV8 & LREQ9)
- 1 frame line (LFRAME10)
- 1 block line (LBLOCK11)

The Local Bus is segmented such that each device with Local Bus capability has two interfaces, one to each of the modules in the physically adjacent slots. Data flows left to right over the Local Bus through adjacent slots. A module may connect the bus segments together, or connect to each segment independently. Figure 4-1 shows how the Memory Module uses the lines.

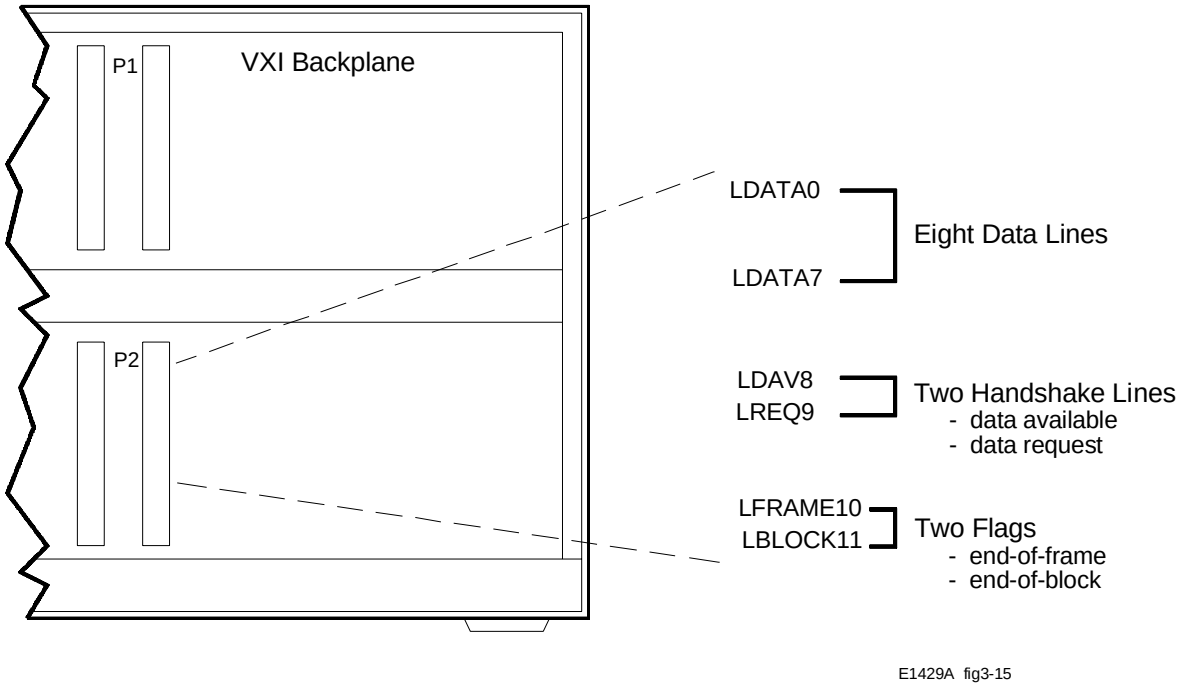


Figure 4-1. Local Bus Signal Lines Definitions

Allowable Signal Levels

The Local Bus supports communications for three signal levels: TTL, ECL, and analog. However, a module may only communicate with other modules that have the same type of signal level (see "Local Bus Key" below). The Memory Module uses the ECL signal level and thus can only transfer the Local Bus to another module that can has the ECL signal level capability. The voltage and current limits of the different levels are:

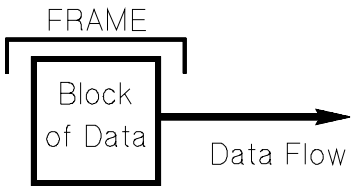
ECL: -5.46V to 0.0V, 50 mA (used by Memory Module)
Analog low: ±5.5V, 50Ω
medium: ±16.0V, 500 mA
high: ±42.0V, 500 mA
Digital TTL: -0.5V to +5.5V, 200 mA

Local Bus Key

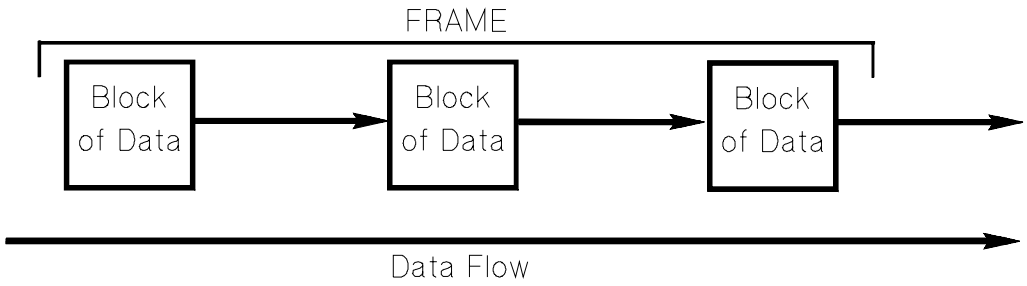
A Local Bus key prevents modules with incompatible Local Bus signal levels from being installed in adjacent slots. The key of the Local Bus module is different if the Local Bus signal levels are different.

How to Transfer Data

Data transferred over the Local Bus is in units of blocks and frames. A frame is a block or a series of blocks where each block is a stream of data bytes output by a single module. Normally, whenever a module outputs data, it also sends an end-of-frame indication (sets (LFRAME10 true) and an end-of-block indication (sets LBLOCK11 true). However, in some cases, the module may only send an end-of-block or an end-of-frame indication (see "Local Bus Operating Modes" below). The following illustrates the relationship of a single block of data in a single frame.



When outputting multiple blocks of data, a single block can be in a single frame or all blocks together can be in a single frame. The following illustrates the relationship of multiple blocks of data in a single frame.



Local Bus Operating Modes

The following lists the Local Bus Operating Modes for the Memory Module. Select the different modes using the command:

VINStrument[:CONFigure]:LBUS[:MODE] <*mode*>

where the <*mode*> parameters are the different operating modes listed below.

APPend: The Memory Module passes Local Bus data through until it detects an end-of-frame indication. It then strips the end-of-frame off and transfers the data in memory out. The module sends an end-of-block and an end-of-frame indication with the last byte of data transferred. Select this mode to append a block of data at the end of all other Local Bus traffic. It is also used for the rightmost Memory Module when using multiple modules to generate one large block of Local Bus data.

CONSume: The Memory Module receives and stores Local Bus data in memory up to the allocated amount of memory. It then disables the Local Bus interface. Select this mode for normal receive-data transfers.

DEFault: Same as CONSume.

EAVesdrop: The Memory Module receives and stores Local Bus data in memory up to the allocated amount of memory but also simultaneously passes it through. It then disables the Local Bus interface. Select this mode to "listen in" on Local Bus traffic.

GENerate: The Memory Module transfers data out of memory to the Local Bus. It sends an end-of-block and an end-of-frame indication with the last byte of data transferred. Select this mode for normal send-data transfers.

INSert: The Memory Module transfers data out of memory to the Local Bus. It sends an end-of-block (but no end-of-frame) indication with the last byte of data transferred. The module then passes Local Bus data through until it detects an end-of-frame indication. Select this mode to add a block of data at the beginning of all Local Bus traffic.

OFF: This disables the Local Bus interface. The Memory Module neither receives or passes Local Bus data through. Select this mode when no Local Bus traffic is anticipated.

MAPPend: The Memory Module passes Local Bus data through until it detects an end-of-frame indication. It then strips the end-of-frame off and transfers the data in memory out. It sends an end-of-frame (but no end-of-block) indication with the last byte of data transferred. Select this mode for the center Memory Module(s) when using multiple modules to generate one large block of Local Bus data.

MGENerate: The Memory Module transfers data out to the Local Bus. It sends an end-of-frame (but no end-of-block) indication with the last byte of data transferred. Select this mode for the leftmost Memory Module when using multiple modules to generate a large amount of Local Bus data.

PIPeline: Select this mode when data should be transparently passed through the Agilent E1488A. The Memory Module stops data transfer until it receives an end-of-frame indication.

STRip: The Memory Module receives and stores Local Bus data in memory up to the allocated amount of memory. It then passes the rest of the Local Bus data through until it receives an end-of-frame indication. Select this mode for all but the rightmost Memory Module when using multiple modules to store a large amount of Local Bus data. The rightmost Memory Module should be set to CONSume mode.

The SCPI Status System

The following describes the SCPI Status System for the Memory Module.

Status System Overview

Figure 4-2 shows the different Status Groups in the Status System. The Questionable Data Group, Operation Status Group, and Standard Event Group can send summary information to the Status Byte. By polling the Status Byte using "*STB?", you can determine if the module generated a condition.

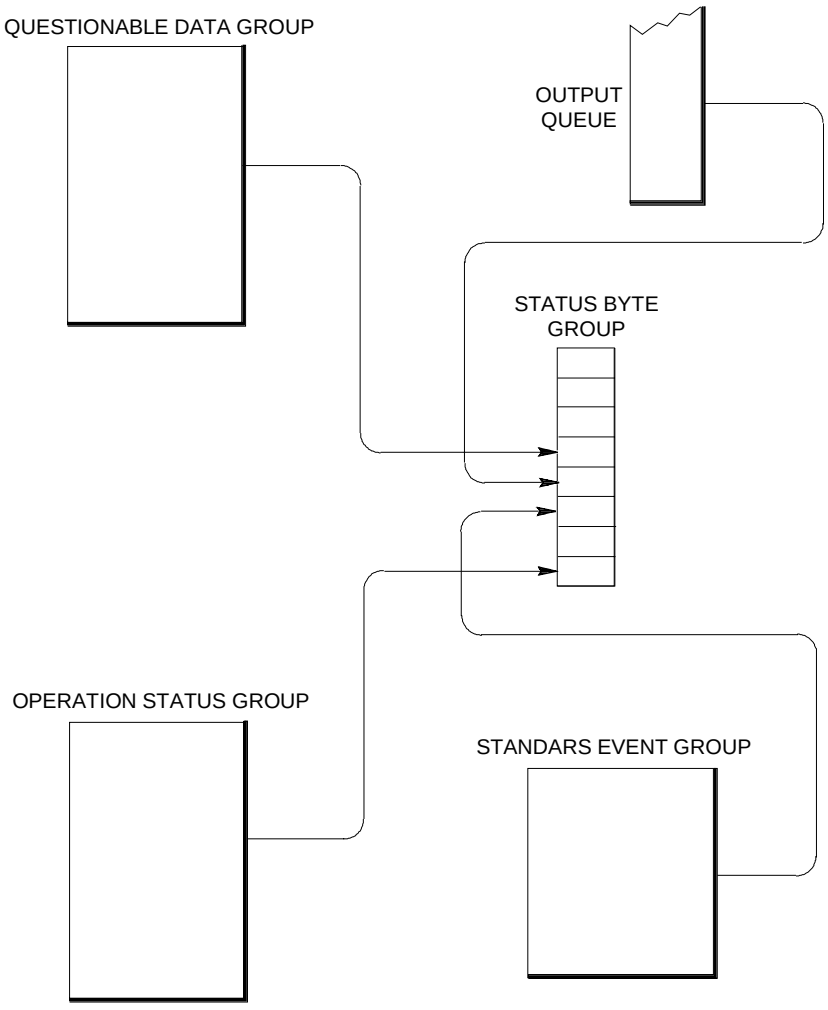


Figure 4-2. Simplified Status System Diagram

The Standard Event Group

The Standard Event Group is shown in Figure 4-3. This group reports the conditions listed in Table 4-1. A condition sets the appropriate bit true (1) in the Event Register. If the corresponding bit in the Enable Register is set true (1), the Standard Event Group sets to true (1) bit 5 in the Summary Register of the Status Byte Group (see "The Status Byte Group" on page 77).

To report the condition in the Status Byte Group, set the enable mask of the Enable Register to the appropriate value. To set the enable mask, sent the decimally weighted sum of the bits you wish to enable with the *ESE (Standard Event Status Enable) common command. For example, to set bits 2 and 5, use a bit value of 4 + 32 = 36. To enable the bits, execute:

```
*ESE 36
```

To clear the Event Register of the Standard Event Group:

- Execute a *CLS (Clear Status) common command.
- Query the Event Register with the *ESR? (Standard Event status Enable Query) common command.

Note An error condition (Event Register bits 2, 3, 4, or 5) always record one or more errors in the module's error queue. Read the error queue using the SYSTem:ERRor? command.

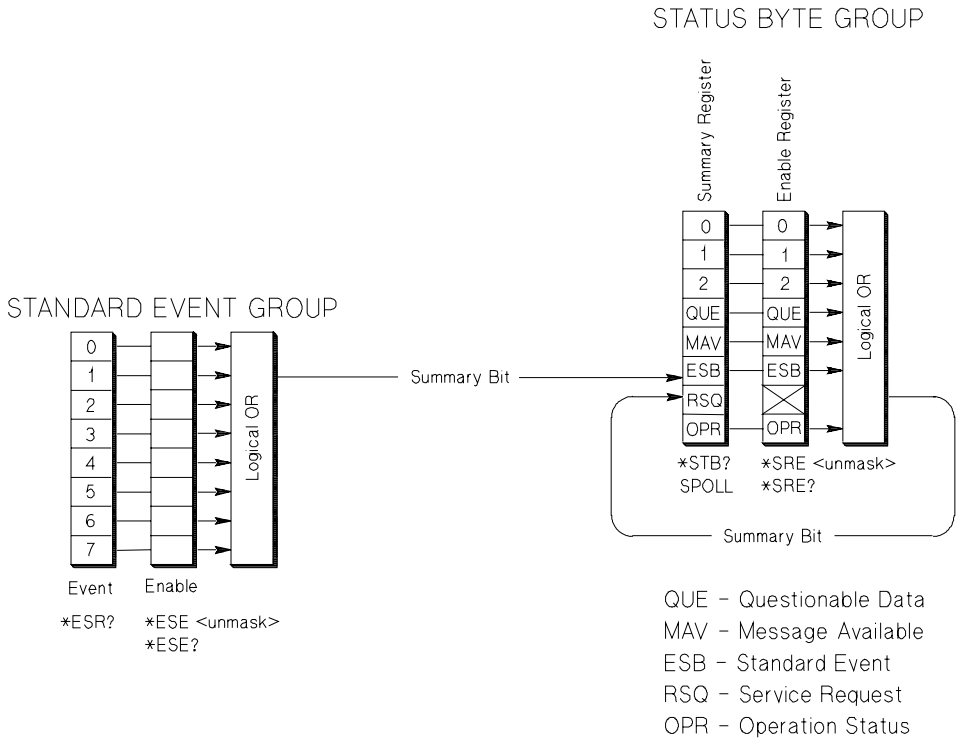


Figure 4-3. Standard Event Group

Table 4-1. Standard Event Group Definitions

Bit Number	Decimal Weight	Description
0	1	**Operation Complete** - shows that the latest operation completed. Bit is set in response to an *OPC command.
1	2	**Request Control** - not functional for this module.
2	4	**Query Error** - an error was noted in the module's output queue.
3	8	Device Dependent error - the module did not complete an operation possibly due to a hardware failure.
4	16	**Execution Erro**r - the module cannot execute the requested operation.
5	32	**Command Error** - the module cannot execute the command.
6	64	**User Request** - the module is undr local control.
7	128	**Power-On** - shows that power-on has occurred.
8		**Not Used** - returns 0.

The Operation Status Group

The Operation Status Group is shown in Figure 4-4. This group only reports the following condition:

Bit 8 - INITiated: Set true (1) by the executing the INITiate[:IMMediate] command.

The condition sets to true (1) bit 8 in the Event Register. If bit 8 in the Enable Register is enabled (1), and bit 8 of the Transition Filter is set for positive (1), the Operation Status Group sets to true (1) bit 7 in the Summary Register of the Status Byte Group (see "The Status Byte Group" on page 77).

To report the condition in the Operation Status Group, set the enable mask of the Enable Register. To set the enable mask, sent the STATus:OPERation|QUEStionable:ENABle command as follows:

```
STATus:OPERation:ENABle  8
```

With the transition filter set positive, any 0 to 1 transition of bit 8 sets bit 7 in the Summary Register of the Status Byte Group to 1. To set the transition to positive, send the following command:

```
STATus:OPERation:PTRansition  8
```

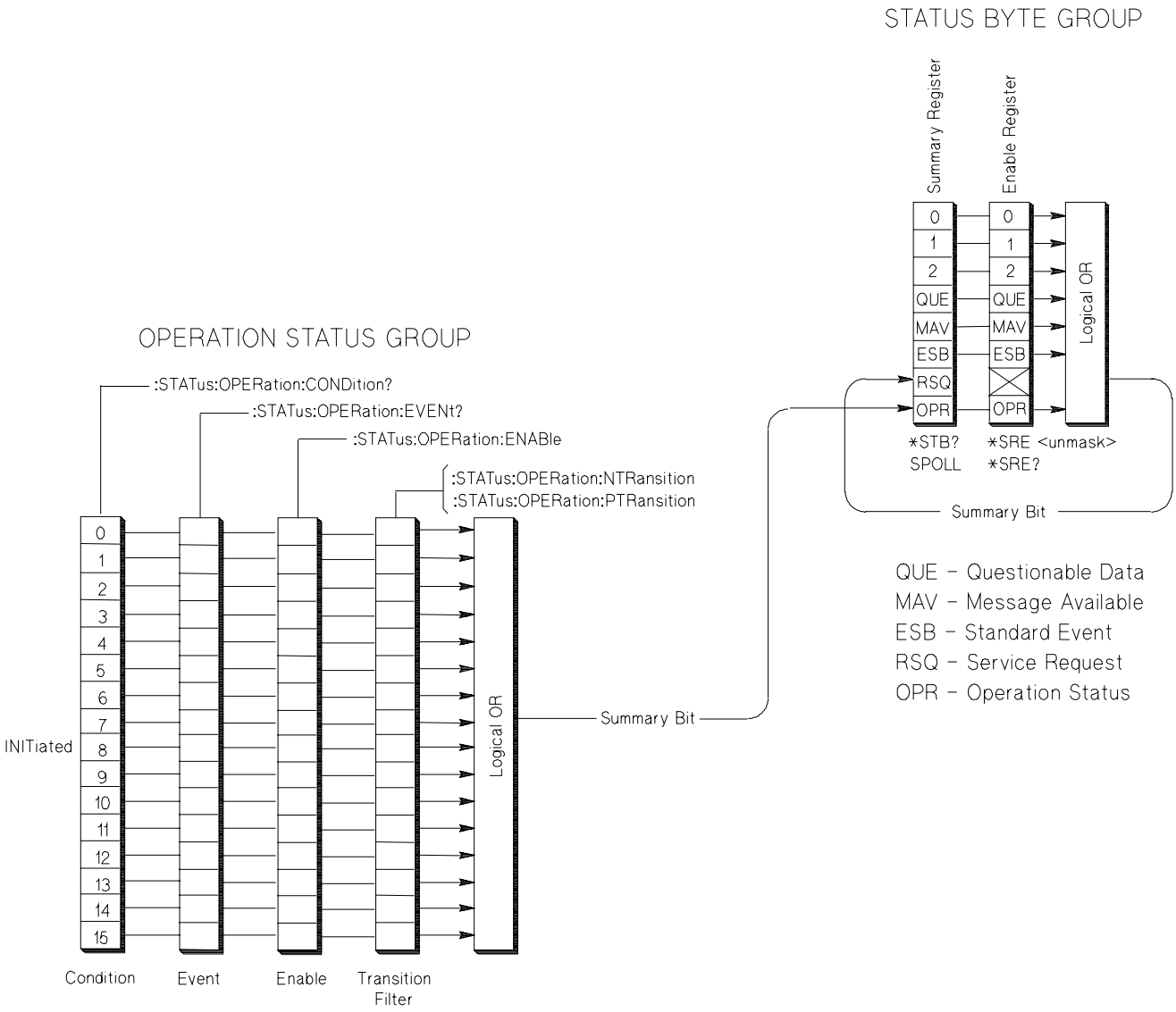


Figure 4-4. Operation Status Group

With the transition filter set to negative, any 1 to 0 transition of bit 8 sets bit 7 in the Summary register of the Status Byte Group to 1. To set the transition to negative, send the following command:

```
STATus:OPERation:NTRansition  8
```

To clear the Event Register in the Operation Status Group, sent the ABORt command. The bit is also cleared after a Local Bus transfer completes.

The Questionable Data Group

This group performs no function in the memory Module. All bits are always 0 in the Event Register of the Questionable Data Group.

The Status Byte Group

The Status Byte Group records high-level summary information reported by the other status groups. If a bit in the Summary Register is set true (1) and the corresponding bit in the Enable register is set true (1), the Status Byte generates a summary bit that sets to true bit 6 (RQS) in the Summary Register.

To report the condition in the Status Byte Group, set the enable mask of the Enable Register to the appropriate value. To set the enable mask, sent the decimally weighted sum of the bits you wish to enable using the *SRE (Service Request Enable Query) common command. For example, to set bits 5 and 7, use a bit value of 32 + 128 = 160. To enable the bits, execute:

```
*SRE  160
```

To clear the Summary Register:

- Execute a *CLS (Clear Status) common command.
- Clear the Event Register in the Standard Event Group and the Condition Register in the Operation Status Group to clear the respective bits in the Summary Register.
- Send the *SRE 0 common command to clear the Enable Register of the Status Byte Group.

Notes

Chapter 5
Command Reference

About this Chapter

this chapter describes the Standard Commands for Programmable Instruments (SCPI) command set and the IEEE 488.2 Common Commands for the Agilent E1488A Instrument Memory Module (Memory Module).

Command Types

Commands are separated into two types: IEEE 488.2 Common Commands and SCPI Commands.

Common Command Format

The IEEE 488.2 standard defines the Common Commands that perform functions like reset, self-test, status byte query, etc. Common Commands are four or five characters in length, always begins with the asterisk character (*), and may include one or more parameters. The command keyword is separated from the first parameter by a space character. Some examples of Common Commands are shown below:

```
*RST, *ESR 32, *STB?
```

SCPI Command Format

The SCPI commands perform functions like enabling the module, receiving data, retrieving data, etc. A SCPI command structure is a hierarchical structure that usually consists of a top level (or root) command, one or more lower level commands, and their parameters. The following example shows part of a typical subsystem:

```
[TRACe | DATA]
    :DELete
        :ALL
        [:NAME]  <name>
```

TRACe or DATA is the root keyword of the command, :DELete is the second level keyword, and :ALL and :NAME are the third level keyword with parameters.

Compound Command Separator

A colon (:) always separates one command from the next lower level command in a compound command, as shown below:

```
TRACe:DELete:ALL
```

Colons separate the root command from the second level command (TRACe:DELete) and the second level from the third level (DELete:ALL).

Abbreviated Commands

The command syntax shows most commands as a mixture of upper and lower case letters. The upper case letters indicate the abbreviated spelling for the command. For shorter program lines, send the abbreviated form. For better program readability, you may send the entire command. The module accepts either the abbreviated form or the entire command.

For example, if the command syntax shows TRACe, then TRACe and TRAC are both acceptable forms. Other form of TRACe, such as TRA or TR generates an error. You may use upper or lower case letters. Therefore, TrAcE, TRACE, and trace are all acceptable.

Implied Commands

Implied Commands are those which appear in square brackets ([ ]) in the command syntax. (Note that the brackets are not part of the command, and are not sent to the module.) Suppose you send a third level command but do not send the preceding implied command. In this case, the module assumes you intend to use the implied command and responds as if you had sent it. Examine part of the TRACe|DATA subsystem shown below:

```
VINStrument
    [:CONFigure]
        :LBUS
            [:MODE] <mode>
            :PAUSed?
            :RESet
```

The second level command [:CONFigure] and the fourth level [:MODE] command are both implied commands. To set the Local Bus mode to CONSume, you could send the following command statements:

```
    VINStrument:CONFigure:LBUS:MODE CONSume
or
    VINStrument:LBUS:MODE CONSume
or
    VINStrument:CONFigure:LBUS CONSume
or
    VINStrument:LBUS CONSume
```

Parameters

Parameter Types. The following table contains explanations sand examples of parameter types shown later in this chapter.

Parameter Types	Explanations and Examples
Numeric	Accepts all commonly used decimal representations of numbers including options signs, decimal points, and scientific notation (NR1, NR2, NR3, and NRf numeric representations as defined in ANSI X3.42-1975 [5]). See the following definitions for NR1, NR2, NR3, and NRf. 123, 123E2, -123, -1.23E2, .123, 1.23E+2, 1.23000E-.01. Special cases include MIN and MAX.
NR1	NR1 Numeric Response Data has the form ±<digits>; for example 123, -123.
NR2	NR2 Numeric Response Data has the form ±<digits>.<digits>; for example 0.123, 12.123.
NR3	NR3 Numeric Response Data has the form ±<digits>.<digits>E±<digits>; for example 123E2, -1.23E2, 1.23E-2, 1.23000E-.01.
NRf	NRf is a flexible version of the three numeric representations (NR1, NR2, or NR3) as defined in ANSI X3.42-1975 [5].
Boolean	Represents a single binary condition that is either true or false. ON, OFF, 1, 0
Discrete	Selects from a finite number of values. These parameters use mnemonics to represent each valid setting. An example is the VINStrument[:CONFigure]:VME[:MODE] command where the source can be CONSume | GENerate | OFF | DEFault
Character Data	12 character maximum, first character must be an alpha character, the character set is alpha-numeric (A-Z, 0-9) and includes the underscore.
Block	A block of of data consisting of a data type specified by NR1. The data can be either bytes or words.

Optional Parameters. :Parameters shown within square brackets ([ ]) are optional parameters. (Note the the brackets are not part of the command, and are not sent to the module.) If you do not specify a value for an optional parameter, the module chooses a default value. A space is required between the command and parameter.

Linking Commands

Linking IEEE 488.2 Common Commands With SCPI Commands

Individual commands can be linked together in a single "program message". These commands will execute, in order, when a "program message terminator" is encountered, i.e. a new-line character, or an EOI or both. Combining commands in this way provides some efficiency improvements for the programmer; a more compact program listing and faster execution. To link IEEE 488.2 Common Commands with SCPI Commands, use a semicolon (;) between the commands. For example,

```
*RST:FORMat:DATA  PACKed
```

LInking Multiple SCPI Commands From Different Subsystems

Use both a semicolon (;) and a colon (:) between the commands. For example:

```
FORMat:DATA  PACKed;:VINStrument:CONFigure:LBUS:RESet
```

Linking Multiple SCPI Commands at the Same Hierarchical Level

Use a semicolon (;) to separate commands and omit the higher level commands from the header. For example, linking VINStrument:CONFigure:LBUS:RESet with VINStrument:CONFigure:LBUS:MODE OFF produces the following shorter form when the two commands are linked:

```
VINStrument:CONFigure:LBUS:RESet;MODE  OFF
```

SCPI Command Reference

The following describes the Standard Commands for Programmable Instruments (SCPI) for the Agilent E1488A Memory Module. Commands are listed alphabetically by subsystem and within a subsystem.

ABORt

The ABORt command places the Local Bus interface in the transfer idle state, regardless of any other settings. The command halts data transfer, but retains any data received prior to executing ABORt. Only another INITiate:IMMediate command will restart data transfer. Any received data should be read before restarting data transfer.

Note After sending ABORt to prematurely halt a Local Bus transfer, the VINStrument:LBUS:RESet command (or equivalent, depending on module type) should be sent all Local Bus modules to the right of the Agilent E1488A, in left-to-right order. This is necessary to ensure proper start-up of the *next* Local Bus transfer.

Subsystem Syntax

ABORt [no query]

Comments

- ABORt does not affect any other settings of the Agilent E1488A, nor does it affect memory contents (TRACe|DATA:DATA).
- The Pending Operation Flag set true by the INITiate:IMMediate command will be set false as a consequence of entering the transfer idle state. Subsequent *OPC, *OPC?, and *WAI commands will therefore complete immediately.
- **Executable when initiated:** Yes
- **Related Commands:** *OPC, *OPC?, *WAI, INITiate:IMMediate, VINStrument:CONFigure:LBUS:RESet
- ***RST Condition:** *RST places the Agilent E1488A in the transfer idle state, as if executing an ABORt command.

Example

Aborting a data transfer

```
ABOR                    Place Agilent E1488A in idle state
```

The DISPlay subsystem enables or disables monitor mode for the external display for the Agilent E1488A.

Subsystem Syntax

```
DISPlay
    :MONitor
        [:STATe] <state>
```

:MONitor[:STATe]

DISPlay:MONitor[:STATe] <***state***> enables or disables monitor mode.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
state	boolean	OFF|0|ON|1	none

Comments

- The monitor mode display for the Agilent E1488A is very limited.
- Pressing any key on the external keyboard will disable monitor mode.
- **Executable when initiated:** Yes
- ***RST Condition:** unaffected
- **Power-On Condition:** DISPlay:MONitor:STATe OFF

Example **Enabling monitor mode**

```
DISP:MON ON                    Enable monitor mode
```

The FORMat subsystem controls the format for return data from the TRACe|DATA:DATA? query.

Subsystem syntax

```
FORMat
    [:DATA]  <format>[,<length>]
```

[:DATA]

FORMat[:DATA] <*format*>[,<*length*>] command specifies the format for return data from the TRACe|DATA:DATA? query. The available return data formats are:

ASCii: Returns numeric data as an NR1 number as defined in IEEE-488.2. Each number represents the value of a memory data byte.

PACKed: Returns data in IEEE-488.2 definite block format. Internal to the block, the bytes are returned in the order received.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
format	discrete	ASCii|PACKed	none
length	numeric	*see below*| MINimum|MAXimum	none
If ASCII format is specified, *length* must either be omitted or must be 9 (or MINimum or MAXimum). Packed format ignores the *length* parameter.			

Comments

- **Executable when initiated:** Yes
- **Related commands:** TRACe|DATA:DATA?
- ***RST Condition:** FORMat:DATA ASCii

Example **Setting PACKed return data format**

```
FORM PACK                    Set packed format
```

The INITiate subsystem starts a Local Bus data transfer.

Subsystem Syntax

```
INITiate
    :CONTinuous <state>
    [:IMMediate]                    [no query]
```

:CONTinuous

INITiate:CONTinuous <***state***> starts a Local Bus data transfer with high-speed restarting after any pauses, when set to ON. Transfers started with INITiate:CONTinuous ON terminate:

by setting INITiate:CONTinuous OFF and executing ABORt, *or*

by setting INITiate:CONTinuous OFF before the normal end of the transfer cycle as described under INITiate:IMMediate. The transfer will terminate as if INITiate:IMMediate had been used to start the transfer.

INITiate:CONTinuous ON is valid only with VINStrument:LBUS:MODE modes APPend, INSert, MAPPend, PIPeline, and STRip.

For transfers that have both a pipeline cycle and a send or receive cycle (APPend, INSert, MAPPend, STRip), the Agilent E1488A, when started by INITiate:IMMediate, pauses for about 250nS after the first cycle before continuing with the second cycle. This delay may be unacceptable in some applications. This delay may be reduced to about 2nS by using INITiate:CONTinuous ON instead of INITiate:IMMediate. However, INITiate:CONTinuous ON usually requires setting INITiate:CONTinuous OFF and ABORt to terminate the Local Bus transfer, and improper operation of the Agilent E1488A will result if multiple Local Bus data frames are transferred.

For pipeline-only transfers, the Agilent E1488A, when started by INITiate:IMMediate, terminates on an end-of-frame indication, and must be restarted by another INITiate:IMMediate. Setting INITiate:CONTinuous ON allows multiple data frames to be transferred automatically and safely, effectively making the Agilent E1488A transparent on the Local Bus.

Setting INITiate:CONTinuous ON is an overlapped command as described by IEEE-488.2, Section 12. The exit from the idle state caused by setting INITiate:CONTinuous ON shall cause its Pending Operation Flag to be set true. This Pending Operation Flag will be set false when the idle state is re-entered. The idle state will be entered momentarily if an ABORt is executed with INITiate:CONTinuous ON set, or normally if INITiate:CONTinuous OFF is set and either the transfer completes or ABORT is executed.

Comments

- Set INITiate:CONTinuous OFF and use the ABORt command to halt the data transfer and place the Agilent E1488A in the idle state.
- **Executable when initiated:** only INITiate:CONTinuous OFF
- **Related Commands:** *OPC, *OPC?, *RST, *WAI, ABORt, INITiate:IMMediate
- ***RST Condition:** The Agilent E1488A is in the idle state.

Example **Initiating a continuous data transfer**

```
INIT:CONT ON                    Initiate data transfer
```

[:IMMediate]

INITiate[:IMMediate] starts a Local Bus data transfer. Send-data transfers complete when all data stored by TRACe|DATA:DATA has been sent. Receive-data transfers complete when the amount of data specified by TRACe|DATA:DEFine is received. Pipeline transfers complete when an end-of-frame indication is received.

For transfers that have both a pipeline cycle and a send or receive cycle (APPend, INSert, MAPPend, STRip), the Agilent E1488A pauses for about 250 nS after the first cycle before continuing with the second cycle. This delay may be unacceptable in some applications. This delay may be reduced to about 2 nS by using INITiate:CONTinuous ON instead of INITiate:IMMediate. However, INITiate:CONTinuous ON usually requires setting INITiate:CONTinuous OFF and ABORt to terminate the Local Bus transfer, and improper operation of the Agilent E1488A will result if multiple Local Bus data frames are transferred.

INITiate:IMMediate is an overlapped command as described by IEEE-488.2, Section 12. The exit from the idle state caused by INITiate:IMMediate shall cause its Pending Operation Flag to be set true. This Pending Operation Flag will be set false when the idle state is re-entered.

Comments

- Use the ABORt command to prematurely halt the data transfer and place the Agilent E1488A in the idle state.
- **Executable when initiated:** No
- **Related Commands:** *OPC, *OPC?, *RST, *WAI, ABORt, INITiate:CONTinuous
- ***RST Condition:** The Agilent E1488A is in the idle state.

Example **Initiating a data transfer**

```
INIT                            Initiate data transfer
```

The MEMory subsystem returns information about memory usage and availability on the Agilent E1488A, and whether memory backup is enabled.

Subsystem syntax

```
[SOURce]
    :MEMory
        :FREE?              [query only]
        :NONVolatile?       [query only]
```

:FREE?

MEMory:FREE? returns information on memory availability and usage. The return data format is:

<numeric_value>,<numeric_value>

The first numeric value shows the amount of unallocated memory; the second, the amount of memory allocated by the TRACe:DEFine command. Both numbers represents bytes. The sum of the two numbers gives the total amount of memory installed on the Agilent E1488A.

Comments

- **Executable when initiated:** Yes
- ***RST Condition:** none
- **Power-On Condition:** all of the waveform segment memory is allocated

Example **Querying memory usage**

```
MEM:FREE?                    Query memory usage
```

NONVolatile?

MEMory:NONVolatile? returns 1 if memory backup has been enabled via the "Backup Enable" switch, or 0 if backup has been disabled. Note that the status of the backup power source is *not* checked, only the state of the "Backup Enable" switch.

Comments

- **Executable when initiated:** Yes
- ***RST Condition:** none

Example **Querying backup enable state**

```
MEM:NONV?                    Query backup enable state
```

The STATus subsystem controls the SCPI-defined Operation and Questionable Signal status registers. Each is comprised of a condition register, an event register, an enable mask, and negative and positive transition filters.

Each status register works as follows: when a condition occurs, the appropriate bit in the condition register is set or cleared. If the the corresponding transition filter is enabled for that bit, the same bit is set in the associated event register. The contents of the event register and the enable mask are logically ANDed bit-for-bit; if any bit of the result is set, the summary bit for that register is set in the status byte. The status byte summary bit for the Operation status register is bit 7; for the Questionable Signal status register, bit 3.

Operation Status Register

Only bit 8 (INITiated) is defined for the Agilent E1488A. All other bits are always zero.

Bit 8 - INITiated: Set (1) by the INITiate:IMMediate command. Cleared (0) by the ABORt command or when the Local Bus transfer completes.

Questionable Signal Status Register

All bits are always 0. This register is implemented only for SCPI compatibility purposes.

Subsystem Syntax

```
STATus
    :OPC
        :INITiate <state>
    :OPERation|QUEStionable
        :CONDition?                    [query only]
        :ENABle <mask>
        [:EVENt]?                      [query only]
        :NTRansition <mask>
        :PTRansition <mask>
    :PRESet                            [no query]
```

:OPC:INITiate

STATus:OPC:INITiate <***state***> controls whether the *OPC, *OPC?, and *WAI commands will complete immediately or whether they will wait for Local Bus transfers to complete. With STATe OFF set, these commands will complete immediately. With STATe ON set, they will wait for the Pending Operation Flag set true by INITiate:IMMediate to return false, indicating that the Local Bus circuitry is in the idle state and that Local Bus transfers have completed or been aborted by the ABORt or *RST commands.

STATus:OPERation|QUEStionable:CONDition?

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
state	boolean	OFF|0|ON|1	none

Comments

- **Executable when initiated:** Yes
- **Coupling group:** none
- **Related commands:** *OPC, *OPC?, *RST, *WAI, ABORt, INITiate:IMMediate, STATus:PRESet
- ***RST Condition:** unaffected
- **Power-on Condition:** STATus:OPC:INITiate ON

Example **Setting immediate completion mode**

```
STAT:OPC:INIT OFF          Complete immediately for *OPC, etc.
```

:OPERation|QUEStionable:CONDition?

STATus:OPERation|QUEStionable:CONDition? returns the contents of the appropriate condition register. Reading the register does not affect its contents.

Comments

- **Executable when initiated:** Yes
- **Coupling group:** none
- **Related commands:** STATus subsystem, *SRE, *STB?
- ***RST Condition:** all bits of both condition registers are cleared as a result of the state present after *RST.

Example **Querying the Operation condition register**

```
STAT:OPER?          Query Operation condition register
```

:OPERation|QUEStionable:ENABle

STATus:OPERation|QUEStionable:ENABle ***<unmask>*** specifies which bits of the associated event register are included in its summary bit. The summary bit is the bit-for-bit logical AND of the event register and the unmasked bit(s).

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
unmask	numeric or non-decimal numeric	0 through +32767	none

The non-decimal numeric forms are the #H, #Q, or #B formats specified by IEEE-488.2.

Comments

- **Executable when initiated:** Yes
- **Coupling group:** none
- **Related commands:** STATus subsystem, *SRE, *STB?
- ***RST Condition:** unaffected
- **Power-on Condition:** STATUS:OPERation|QUEStionable:ENABLE 0

Example **Setting the Operation register enable mask**

```
STAT:OPER:ENAB #H0100          Enable summary on Initiated bit
```

:OPERation|QUEStionable[:EVENt]?

STATus:OPERation|QUEStionable[:EVENt]? returns the contents of the appropriate event register. Reading the register clears it to 0.

Comments

- Both event registers are also cleared to 0 by the *CLS common command.
- **Executable when initiated:** Yes
- **Coupling group:** none
- **Related commands:** STATus subsystem, *SRE, *STB?
- ***RST Condition:** unaffected
- **Power-on Condition:** Both event registers are cleared to 0.

Example **Querying the Operation event register**

```
STAT:EVEN?                     Query Operation event register
```

:OPERation|QUEStionable:NTRansition

STATus:OPERation|QUEStionable:NTRansition <***unmask***> sets the negative transition mask. For each bit unmasked, a 1-to-0 transition of that bit in the associated condition register will set the same bit in the associated event register.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
unmask	numeric or non-decimal numeric	0 through +32767	none

The non-decimal numeric forms are the #H, #Q, or #B formats specified by IEEE-488.2.

Comments

- **Executable when initiated:** Yes
- **Coupling group:** none
- **Related commands:** STATus subsystem, *SRE, *STB?
- ***RST Condition:** unaffected
- **Power-on Condition:** STATUS:OPERation|QUEStionable:NTRansition 0

Example

Setting the Operation register negative transition mask

```
STAT:OPER:NTR #H0100          Set event bit when transfer complete
```

:OPERation|QUEStionable:PTRansition

STATus:OPERation|QUEStionable:PTRansition <***unmask***> sets the positive transition mask. For each bit unmasked, a 0-to-1 transition of that bit in the associated condition register will set the same bit in the associated event register.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
unmask	numeric or non-decimal numeric	0 through +32767	none

The non-decimal numeric forms are the #H, #Q, or #B formats specified by IEEE-488.2.

Comments

- **Executable when initiated:** Yes
- **Coupling group:** none

- **Related commands:** STATus subsystem, *SRE, *STB?
- ***RST Condition:** unaffected
- **Power-on Condition:** STATUS:OPERation|QUEStionable:PTRansition 32767

Example **Setting the Operation register positive transition mask**

```
STAT:OPER:PTR #H0100          Set event bit when transfer started
```

:PRESet

STATus:PRESet initializes the enable registers and transition masks for the Operation and Questionable Signal status registers and sets STATus:OPC:INITiate ON. For both status registers, the enable registers are set to 0, the negative transition masks are set to 0, and the positive transition masks are set to 32767.

Comments

- **Executable when initiated:** Yes
- **Coupling group:** none
- **Related commands:** STATus subsystem, *SRE, *STB?
- ***RST Condition:** none

Example **Presetting the STATus subsystem**

```
STAT:PRES                     Preset STATus subsystem
```

The SYSTem subsystem returns error messages and the SCPI version number to which the Agilent E1488A complies.

Subsystem Syntax

```
SYSTem
    :ERRor?          [query only]
    :VERSion?        [query only]
```

:ERRor?

SYSTem:ERROR? returns the error messages in the error queue. See Appendix B for a listing of possible error numbers and messages.

Comments

- The Agilent E1488A places any generated errors into the error queue. The queue is first-in, first out. With several errors waiting in the queue, the SYSTem:ERRor? returns the oldest unread error message first.
- The error queue can hold 30 error messages. If the Agilent E1488A generates more than 30 messages that are not read, it replaces the last error message in the queue with error -350,"Too many errors". No additional messages are placed into the queue until SYSTem:ERRor? reads some messages or the *CLS (clear status) command clears the queue.
- When the error queue is empty, SYSTem:ERRor? returns +0,"No error".
- **Executable when initiated:** Yes
- ***RST Condition:** unaffected
- **Power-On Condition:** no errors are in the error queue

Example **Reading the error queue**

```
SYST:ERR?                    Query the error queue
```

:VERSion?

SYSTem:VERSion? returns the SCPI version number to which the Agilent E1488A complies: "1992.0".

Comment

- **Executable when initiated:** Yes
- ***RST Condition:** none

Example **Querying the SCPI revision**

```
SYST:VERS?                    Query SCPI revision
```

The TRACe|DATA subsystem allocates memory and transfers data to/from memory. Either TRACe or DATA may be used as the root node.

Subsystem syntax

```
TRACe|DATA
    :CATalog?                                   [query only]
    [:DATA] <name>,<data_list>
        :COUNt <name>,<length>
        :PARtial? <name,<start>,<stop>          [query only]
    :DEFine <name>,<length>
    :DELete
        :ALL                                    [no query]
        [:NAME] <name>                          [no query]
```

:CATalog?

TRACe|DATA:CATalog? returns the name of any defined trace, or a single null string ("") if no trace is defined.

Comments

- **Executable when initiated:** Yes
- ***RST Condition:** none
- **Power-On Condition:** no trace is defined

Example **Cataloging trace names**

```
TRAC:CAT?                    Catalog traces
```

[:DATA]

TRACe|DATA[:DATA] ***<name>,<data_list>*** stores data into the Agilent E1488A's memory. The query form, **TRACe|DATA[:DATA]?** ***<name>***, reads data from the memory.

Parameters *Name* is name of the trace to be stored or read. It must have been previously allocated by the TRACe|DATA:DEFine command, and, for the query form, must contain data.

The *data_list* may be either a comma-separated list of byte values or an IEEE-488.2 definite or indefinite length block containing the byte values in 8-bit integer format.

Comments

- If the comma-separated list of values format is used, the values may be in either unsigned or two's complement format, ie., values may range from -128 to +255.
- The data list must be any size up to the allocated length specified by TRACe|DATA:DEFine. If the data list length is less than the allocated length,

only the number of bytes specified by the data list is sent as part of a Local Bus send-data transfer, or is returned by the TRACe|DATA:DATA? query.

- Use the TRACe|DATA:DATA:PARTial? query to return only a portion of trace contents.
- **Executable when initiated:** No
- **Related Commands:** TRACe|DATA:DEFine, TRACe|DATA:DATA:PARTial
- ***RST Condition:** unaffected
- **Power-On Condition:** no trace is defined

Example

Storing data

```
TRAC:DEF ABC,8                  ABC is 8 bytes long
DATA ABC,#15HELLO               Store "HELLO" as data
```

[:DATA]:COUNt

TRACe|DATA[:DATA]:COUNt ***<name>,<length>*** overrides the amount of data stored and sets the amount to the number of bytes specified. This is useful primarily when recovering from a power failure. The Agilent E1488A, if so enabled, will maintain memory contents during a power failure; however, the current length information is not maintained. After power is restored, this command may be used to restore the length.

The query form,**TRACe|DATA[:DATA]:COUNt? *<name>*,** returns the amount of data currently stored in bytes.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
name	character data	existing trace name	none
length	numeric	*see below*| MINimum|MAXimum	none
length must be between 1 and the allocated length of the trace, inclusive. MINimum selects byte 1; MAXimum selects the allocated length.			

Comments

- **Executable when initiated:** No
- ***RST Condition:** none
- **Power-On Condition:** no trace is defined

Example **Querying stored data size**

TRAC:COUN? *Query stored data size*

[:DATA]:PARTial?

TRACe|DATA[:DATA]:PARTial? ***<name>,<start>,<stop>*** returns that portion of the data stored in the Agilent E1488A's memory starting with byte number *start* and ending with byte number *stop*, inclusive. This command is useful when only part of a large trace is wanted, or when the external controller's memory is too small to hold the entire trace at once.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
name	character data	existing trace name	none
start	numeric	*see below*| MINimum|MAXimum	none
stop	numeric	*see below*| MINimum|MAXimum	none
Start must be between 1 and the number of bytes stored, inclusive. MINimum selects byte 1; MAXimum selects the last byte stored. *Stop* must be between the *start* byte and the number of bytes stored, inclusive. MINimum selects the *start* 1; MAXimum selects the last byte stored.			

Comments

- **Executable when initiated:** No
- **Related Commands:** TRACe|DATA:DEFine, TRACe|DATA:DATA
- ***RST Condition:** unaffected
- **Power-On Condition:** no trace is defined

Example **Query part of data**

TRAC:PART? ABC,11,20 *Read bytes 11 to 20 of ABC*

:DEFine

TRACe|DATA:DEFine ***<name>,<length>*** creates a trace with the specified name and a length of *length* bytes. Only one trace may exist at any time; TRACe|DATA:DEFine will generate error +1002, "Trace already defined" if a trace already exists.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
name	character data	1 through 12 characters	none
length	numeric	*see below*| MINimum|MAXimum	none
The *length* may be any value from 4 bytes to all memory installed for Local Bus send-data transfers. It may be any value from 16 bytes to all memory installed for Local Bus receive-data transfers. The length will be rounded up to a multiple of 4 bytes. MINimum allocates 16 bytes if VINStrument:LBUS:MODE CONsume|EAVesdrop|STRip is set, 4 bytes otherwise; MAXimum allocates all memory installed.			

Comments

- Once a trace has been DEFined, it must be deleted (TRACe|DATA:DELete command) before its allocated length may be redefined. The data list values and length may be changed repeatedly without re-executing the DEFine command.
- TRACe|DATA:DEFine initializes the trace's data list to zero length. Actual memory contents are not initialized.
- **Executable when initiated:** No
- **Related Commands:** TRACe|DATA:DATA, TRACe|DATA:DELete
- ***RST Condition:** unaffected
- **Power-On Condition:** no trace is defined

Example **Allocating memory**

```
TRAC:DEF ABC,1024          Create ABC with length 1024 bytes
```

:DELete:ALL

TRACe|DATA:DELete:ALL deletes any defined trace definition from memory, regardless of name.

Comments

- Actual memory contents are unaffected.
- Use TRACe|DATA:DELete:NAME to delete a defined trace while verifying that the named trace exists.
- **Executable when initiated:** No
- **Coupling group:** none
- **Related Commands:** TRACe|DATA:DELete:NAME

- ***RST Condition:** none
- **Power-On Condition:** no trace is defined

Example **Deleting any trace definition**

```
TRAC:DEL:ALL                    Delete trace definition
```

:DELete[:NAME]

TRACe|DATA:DELete[:NAME] <***name***> deletes the specified trace name. The specified trace name must have been previously defined.

Comments

- Actual memory contents are unaffected.
- Use TRACe|DATA:DELete:ALL to delete any trace definition regardless of name.
- **Executable when initiated:** No
- **Related Commands:** TRACe|DATA:DELete:ALL
- ***RST Condition:** none
- **Power-On Condition:** no trace is defined

Example **Deleting a trace**

```
TRAC:DEL ABC                    Delete trace ABC
```

The VINSTrument (Virtual INStrument) subsystem controls the operating mode (send, receive, etc.) of the Local Bus and VME Data Transfer Bus interfaces, permits querying of the completion status of a transfer, and provides other information about the Local Bus and VME Data Transfer Bus implementation of the Agilent E1488A.

Some of the commands in this subsystem are included only for compatibility with the Agilent Virtual Instrument/Local Bus Specification, and are not normally used; this is indicated in the command descriptions. The important commands in this subsystem are the VINStrument[:CONFigure]:LBUS:MODE and VINStrument[:CONFigure]:VME:MODE commands. Also, the VINSTrument:PAUSed? query is important in some operating modes.

Subsystem Syntax

```
VINStrument
    [:CONFigure]
        :LBUS
            [:MODE] <mode>
            :PAUSed?                        [query only]
            :RESet                          [no query]
        :TEST
            :DATA <block>                   [no query]
            :DATA? <length>                 [query only]
        :VME
            [:MODE] <mode>
            :RECeive
                :ADDRess
                    :DATA?                  [query only]
                    :READy?                 [query only]
            :SEND
                :ADDRess
                    :DATA?                  [query only]
                    :READy?                 [query only]
    :IDENtity?                              [query only]
```

[:CONFigure]:LBUS[:MODE]

VINStrument[:CONFigure]:LBUS[:MODE] <***mode***> selects the operating mode for the VXIbus Local Bus. The available modes are:

APPend: The Memory Module passes Local Bus data through until it detects an end-of-frame indication. It then strips the end-of-frame off and transfers the data in memory out. The module sends an end-of-block and an end-of-frame indication with the last byte of data transferred. Select this mode to append a block of data at the end of all other Local Bus traffic. It is also used for the rightmost Memory Module when using multiple modules to generate one large block of Local Bus data.

CONSume: The Memory Module receives and stores Local Bus data in memory up to the allocated amount of memory. It then disables the Local Bus interface. Select this mode for normal receive-data transfers.

DEFault: Same as CONSume.

EAVesdrop: The Memory Module receives and stores Local Bus data in memory up to the allocated amount of memory but also simultaneously passes it through. It then disables the Local Bus interface. Select this mode to "listen in" on Local Bus traffic.

GENerate: The Memory Module transfers data out of memory to the Local Bus. It sends an end-of-block and an end-of-frame indication with the last byte of data transferred. Select this mode for normal send-data transfers.

INSert: The Memory Module transfers data out of memory to the Local Bus. It sends an end-of-block (but no end-of-frame) indication with the last byte of data transferred. The module then passes Local Bus data through until it detects an end-of-frame indication. Select this mode to add a block of data at the beginning of all Local Bus traffic.

OFF: This disables the Local Bus interface. The Memory Module neither receives or passes Local Bus data through. Select this mode when no Local Bus traffic is anticipated.

MAPPend: The Memory Module passes Local Bus data through until it detects an end-of-frame indication. It then strips the end-of-frame off and transfers the data in memory out. It sends an end-of-frame (but no end-of-block) indication with the last byte of data transferred. Select this mode for the center Memory Module(s) when using multiple modules to generate one large block of Local Bus data.

MGENerate: The Memory Module transfers data out to the Local Bus. It sends an end-of-frame (but no end-of-block) indication with the last byte of data transferred. Select this mode for the leftmost Memory Module when using multiple modules to generate a large amount of Local Bus data.

PIPeline: Select this mode when data should be transparently passed through the Agilent E1488A. The Memory Module stops data transfer until it receives an end-of-frame indication.

STRip: The Memory Module receives and stores Local Bus data in memory up to the allocated amount of memory. It then passes the rest of the Local Bus data through until it receives an end-of-frame indication. Select this mode for all but the rightmost Memory Module when using multiple modules to store a large amount of Local Bus data. The rightmost Memory Module should be set to CONSume mode.

Note After changing the Local Bus operating mode, the VINStrument:LBUS:RESet command (or equivalent, depending on module type) *must* be sent all Local Bus modules to the right of the Agilent E1488A, in left-to-right order. This is necessary to ensure proper start-up of the *next* Local Bus transfer.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
mode	discrete	APPend|CONSume|DEFault| EAVesdrop|GENerate|INSert| MAPPend|MGENerate| OFF|PIPeline|STRip	none

Comments

- **Executable when initiated:** No
- **Related Commands:** INITiate:IMMediate, VINStrument:CONFigure:LBUS:RESet
- ***RST Condition:** VINStrument[:CONFigure]:LBUS:MODE OFF

Example **Setting the Local Bus operation mode**

```
VINS:LBUS PIP                    Set pipeline (pass through) mode
```

[:CONFigure]:LBUS:PAUSed?

VINStrument[:CONFigure]:LBUS:PAUSed? returns a value of 1 if the Local Bus circuitry is in the Paused or inactive state or 0 if a Local Bus transfer is in progress (send, receive, or pipeline).

Comments

- **Executable when initiated:** Yes
- ***RST Condition:** none

Example **Querying paused status**

```
VINS:LBUS:PAUS?                  Query paused status
```

[:CONFigure]:LBUS:RESet

VINStrument[:CONFigure]:LBUS:RESet resets the Local Bus Controller chip.

It is good programming practice to send this command before starting every Local Bus transfer, but is *necessary* after changing the Local Bus operating mode (VINStrument:CONFigure:LBUS:MODE command), after aborting a Local Bus transfer (ABORt command), and after resetting the Local Bus Controller chip of any Local Bus modules to the left of the Agilent E1488A.

Note After resetting the Controller chip, all Local Bus modules to the right of the Agilent E1488A, in left-to-right order, *must* also be reset. This is necessary to ensure proper start-up of the *next* Local Bus transfer.

Comments

- **Executable when initiated:** No
- **Related Commands:** *RST, *RCL, ABORt, VINStrument:CONFigure:LBUS:MODE
- ***RST Condition:** none
- **Power-on Condition:** the Local Bus Controller chip is reset

Example **Resetting the Local Bus Controller chip**

```
VINS:LBUS:RES                    Reset Controller chip
```

[:CONFigure]:TEST:DATA

VINStrument[:CONFigure]:TEST:DATA <***block***> configures the Agilent E1488A to generate data for Local Bus testing. The data is stored in the memory and a send-data transfer is configured and initiated.

Parameters The *block* is an IEEE-488.2 definite or indefinite length block containing the test data byte values in 8-bit integer format. The block length must be at least 16 bytes and must be a multiple of 4 bytes.

Note The test destroys any data in memory (TRACe|DATA:DATA). For this reason, this command cannot be executed when any trace is defined. Use the TRACe|DATA:DELete commands to delete a trace definition before executing this command.

Comments

- **Executable when initiated:** No
- **Related Commands:** TRACe|DATA:DELete subsystem, VINStrument[:CONFigure]:TEST:DATA?
- ***RST Condition:** none
- **Power-On Condition:** Local Bus testing not configured

Example **Testing Local Bus operation**

```
VINS:TEST:DATA #216ABCDEFGHIJKLMNOP
```

Send 16 bytes as test

[:CONFigure]:TEST:DATA?

VINStrument[:CONFigure]:TEST:DATA? **<*length*>** configures the Agilent E1488A to receive data for Local Bus testing. A receive-data transfer is configured and initiated. When the transfer completes, the data received is returned in 8-bit integer format in an IEEE-488.2 definite block.

Note The test destroys any data in memory (TRACe|DATA:DATA). For this reason, this command cannot be executed when any trace is defined. Use the TRACe|DATA:DELete:NAME|ALL commands to delete a trace definition before executing this command.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
length	numeric	*see below*| MINimum|MAXimum	none
length must be between 16 and all memory installed. MINimum selects 16 bytes; MAXimum selects all memory installed.			

Comments

- **Executable when initiated:** No
- **Related Commands:** TRACe|DATA:DELete subsystem, VINStrument[:CONFigure]:TEST:DATA
- ***RST Condition:** none
- **Power-On Condition:** Local Bus testing not configured

Example **Testing Local Bus operation**

```
VINS:TEST:DATA? 16          Receive 16 bytes as test
```

[:CONFigure]:VME[:MODE]

VINStrument[:CONFigure]:VME[:MODE] <***mode***> prepares the Agilent E1488A for **register-level** VME data bus transfers. This command is not used when using the TRACe|DATA:DATA command and query.

The available modes are:

CONSume: Prepares for register-level write access to the memory from the VME data transfer bus. Select this mode to load memory at register level before a send-data transfer. Subsequent writes to the Receive Data MSW/LSW registers will write to memory starting at the beginning of memory. Mode OFF must be set after all data has been sent.

After setting mode OFF, the amount of data written will automatically be determined, and the TRACe|DATA:COUNt value updated. However, this length will always be a multiple of 4 bytes. If a different length is desired, the TRACe|DATA:COUNt command may be used to set that length.

DEFault: Same as OFF.

GENerate: Prepares for register-level read access to the memory from the VME data transfer bus. Select this mode to read memory at register level after a receive-data transfer. Subsequent reads from the Send Data MSW/LSW registers will read memory starting at the beginning of memory. Mode OFF must be set after all data has been read.

OFF: Indicates that a register-level VME data transfer bus access to the memory has been completed.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
mode	discrete	CONSume|DEFault| GENerate|OFF	none

Comments

- **Executable when initiated:** No
- **Related Commands:** TRACe|DATA:DATA, TRACe|DATA:COUNt
- ***RST Condition:** VINStrument[:CONFigure]:VME:MODE OFF

Example **Enabling the VME data transfer bus write access**

```
TRAC:DEF ABC,1e6                    Allocate 1 MByte of memory
VINS:VME CONS                       Prepare to load memory from register level
write data to Receive Data MSW/LSW registers
VINS:VME OFF                        Indicate all data written
```

[:CONFigure]:VME:RECeive:ADDRess:DATA?

VINStrument[:CONFigure]:VME:RECeive:ADDRess:DATA? returns two values: A16,32. These values indicate that register offset 20_{16} in the Agilent E1488A's A16 address space should be used for memory data, which corresponds to the Receive Data MSW register.

Comments

- **Executable when initiated:** Yes
- ***RST Condition:** none

Example **Querying the receive address**

```
VINS:VME:REC:ADDR:DATA?             Query address for data writes
```

[:CONFigure]:VME:RECeive:ADDRess:READy?

VINStrument[:CONFigure]:VME:RECeive:ADDRess:READy? returns two values: A16,38. These values indicate that register offset 26_{16} in the Agilent E1488A's A16 address space should be used for the ready indication, which corresponds to the Receive Status register.

Comments

- **Executable when initiated:** Yes
- ***RST Condition:** none

Example **Querying the ready indication address**

```
VINS:VME:REC:ADDR:READ?             Query ready indication address
```

[:CONFigure]:VME:SEND:ADDRess:DATA?

VINStrument[:CONFigure]:VME:SEND:ADDRess:DATA? returns two values: A16,32. These values indicate that register offset 20_{16} in the Agilent E1488A's A16 address space should be used for memory data, which corresponds to the Send Data MSW register.

Comments

- **Executable when initiated:** Yes
- ***RST Condition:** none

Example **Querying the receive address**

```
VINS:VME:REC:ADDR:DATA?          Query address for data writes
```

[:CONFigure]:VME:SEND:ADDRess:READy?

VINStrument[:CONFigure]:VME:SEND:ADDRess:READy? returns two values: A16,36. These values indicate that register offset 24_{16} in the Agilent E1488A's A16 address space should be used for the ready indication, which corresponds to the Send Status register.

Comments

- **Executable when initiated:** Yes
- ***RST Condition:** none

Example **Querying the ready indication address**

```
VINS:VME:REC:ADDR:READ?          Query ready indication address
```

:IDENtity?

VINStrument:IDENtity? returns a response consisting of 4 fields, indicating the virtual instrument capability of the Agilent E1488A:

```
HEWLETT-PACKARD VIRTUAL INSTRUMENT,ANY  STOR,0,A.01.00
```

The first and last fields indicate that the Agilent E1488A conforms to revision A.01.00 of Agilent's Virtual Instrument/Local Bus System Specification. The second field indicates that the Agilent E1488A is a data storage module. The third field is reserved for future use.

Comments

- **Executable when initiated:** Yes
- ***RST Condition:** none

Example **Querying virtual instrument capability**

```
VINS:IDEN?                       Query capability
```

IEEE-488.2 Common Commands

This section describes the IEEE-488.2 Common Commands implemented in the Agilent E1488A. The table below shows the commands listed by functional group; however, commands are listed alphabetically in the reference. Examples are shown in the reference when the command has parameters or returns a non-trivial response; otherwise, the command string is as shown in the table. For additional information, refer to IEEE Standard 488.2-1987.

Category	**Command**	**Title**
System Data	*IDN?	Identification Query
Internal Operations	*LRN? *RST *TST	Learn Device Setup Query Reset Command Self Test Query
Synchronization	*OPC *OPC? *WAI	Operation Complete Command Operation Complete Command Wait-to-Continue Command
Macro	*DMC <*name*>,<*data*> *EMC <*enable*> *EMC? *GMC? <*name*> *LMC? *PMC *RMC <*name*>	Define Macro Command Enable Macro Command Enable Macro Query Get Macro Contents Query Learn Macro Query Purge Macros Command Remove Individual Macro Command
Status & Event	*CLS *ESE <*mask*> *ESE? *ESR? *SRE <*mask*> *SRE? *STB?	Clear Status Command Standard Event Status Enable Command Standard Event Status Enable Query Standard Event Status Register Query Service Request Enable Command Service Request Enable Query Read Status Byte Query
Stored Settings	*RCL *SAV	Recall Command Save Command

*CLS

***CLS** clears the Standard Event Status Register, the Operation Status Register, the Questionable Signal Register, and the error queue. This clears the corresponding summary bits (3, 5, & 7) in the Status Byte Register. *CLS does not affect the enable masks of any of the status registers.

Comments

- **Executable when initiated:** Yes
- **Related Commands:** STATus:PRESet
- ***RST Condition:** none

*DMC

DMC** ***<name>,<data> creates a macro with the specified name and assigns zero, one, or a sequence of commands to the name. The sequence may be composed of SCPI and/or Common Commands. The sequence may be sent in IEEE-488.2 definite or indefinite block format or as a quoted string.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
name	string	1 through 12 characters	none
data	block data or string	any valid command sequence	none

Comments

- Legal macro names must start with an alphabetic character and contain only alphabetic, numeric, and underscore ("_") characters. Alphabetic character case (upper vs. lower) is ignored.

  The name is allowed to be the same as a SCPI command, but may not be the same as a Common Command. When the name is the same as a SCPI command, the macro rather than the command will be executed when the name is received if macro usage is enabled. The SCPI command will be executed if macro usage is disabled.

- **Executable when initiated:** Yes
- **Related Commands:** *EMC, *GMC, *LMC, *RMC
- ***RST Condition:** none; macro defintions are unaffected
- **Power-On Condition:** no macros are defined

Example

Define macro to restart data transfer

```
*DMC "RESTART","ABOR;INIT"          Define macro
```

*EMC and *EMC?

***EMC <_enable_>** enables and disables macro usage. When *enable* is zero, macros usage is disabled. Any non-zero value in the range of -32768 through +32767 enables macro usage.

The query form returns 1 if macro usage is enabled, 0 if disabled.

Comments

- Macro definitions are not affected by this command.
- **Executable when initiated:** Yes
- ***RST Condition:** macro usage is disabled
- **Power-On Condition:** macro usage is enabled

*ESE and *ESE?

***ESE <_mask_>** enables one or more event bits of the Standard Event Status Register to be reported in bit 5 (the Standard Event Status Summary Bit) of the Status Byte Register. *Mask* is the sum of the decimal weights of the bits to be enabled.

The query form returns the current enable mask.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
mask	numeric	0 through 255	none

A 1 in a bit position enables the corresponding event; a 0 disables it.

Comments

- **Executable when initiated:** Yes
- **Related Commands:** *ESR?, *SRE, *STB?
- ***RST Condition:** unaffected
- **Power-On Condition:** no events are enabled

Example

Enable all error events

```
*ESE 60                    Enable error events
```

*ESR?

***ESR?** returns the value of the Standard Event Status Register. The register is then cleared (all bits 0).

Comments

- **Executable when initiated:** Yes
- ***RST Condition:** none
- **Power-On Condition:** register is cleared

*GMC?

***GMC? <*name*>** returns the definition of the specified macro in IEEE-488.2 definite block format.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
name	character data	*defined macro name*	none

Comments

- **Executable when initiated:** Yes
- **Related Commands:** *DMC
- ***RST Condition:** none
- **Power-On Condition:** no macros are defined

Example

Query macro definition

```
*GMC? RESTART                    Query macro definition
```

*IDN?

***IDN?** returns indentification information for the Agilent E1488A. The response consists of four fields:

```
HEWLETT-PACKARD,E1488A,0,A.01.00
```

The first two fields identify this instrument as model number E1488A manufactured by Agilent Technologies. The third field is 0 since the serial number of the Agilent E1488A is unknown to the firmware. The last field indicates the revision level of the firmware.

Note The firmware revision field will change whenever the firmware is revised. A.01.00 is the initial revision. The first two digits indicate the major revision number, and increment when functional changes are made. The last two digits indicate bug fix level.

Comments

- **Executable when initiated:** Yes
- ***RST Condition:** none
- **Power-On Condition:** register is cleared

*LMC?

***LMC?** returns a comma-separated list of quoted strings, each containing the name of a macro. If no macros are defined, a single null string ("") is returned.

Comments

- **Executable when initiated:** Yes
- **Related Commands:** *DMC
- ***RST Condition:** none
- **Power-On Condition:** no macros are defined

*LRN?

***LRN?** returns a sequence of commands that may be resent to the Agilent E1488A to return it to its current programming state.

Only those commands that are affected by *RST are included in the sequence. Notable exceptions include memory contents (TRACe|DATA:DATA) and the STATus subsystem.

Note *LRN? should be sent singly in a program message, since the number of commands in the returned sequence is large, and may vary depending on firmware revision.

Comments

- **Executable when initiated:** Yes
- **Related commands:** *RCL, *RST, *SAV
- ***RST Condition:** none

*OPC

***OPC** causes the Agilent E1488A to wait for all pending operations to complete. The Operation Complete bit (bit 0) in the Standard Event Status Register is then set.

If STATus:OPC:INITiate OFF is set, the Operation Complete bit will be set when all commands received prior to the *OPC have been executed. If ON is set, *OPC waits for data transfers to complete before setting the Operation Complete bit. No other commands will be executed until the Operation Complete bit is set.

Comments

- **Executable when initiated:** Yes
- **Related commands:** *OPC?, *WAI
- ***RST Condition:** none

*OPC?

***OPC?** causes the Agilent E1488A to wait for all pending operations to complete. A single ASCII "1" is then placed in the output queue.

If STATus:OPC:INITiate OFF is set, the ASCII "1" will be placed in the output queue when all commands received prior to the *OPC? have been executed. If ON is set, *OPC? waits for data transfers to complete before placing the "1" in the output queue. No other commands will be executed until the "1" is placed in the output queue.

Comments

- **Executable when initiated:** Yes
- **Related commands:** *OPC, *WAI
- ***RST Condition:** none

*PMC

***PMC** purges all macro definitions.

Comments

- Use the *RMC command to purge a single macro definition.
- **Executable when initiated:** Yes
- **Related commands:** *DMC, *RMC
- ***RST Condition:** none

*RCL

RCL** <number***> restores a previously stored programming state from one of the 10 possible stored state areas. *Number* indicates which of the stored state areas should be used.

This command affects the same command settings as does *RST. Notable exceptions include memory contents (TRACe|DATA:DATA) and the STATus subsystem.

Note

After sending a *RCL that changes the Local Bus operating mode, the VINStrument:LBUS:RESet command (or equivalent, depending on module type) *must* be sent all Local Bus modules to the right of the Agilent E1488A, in left-to-right order. This is necessary to ensure proper start-up of the *next* Local Bus transfer.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
number	numeric	0 through 9	none

Comments

- **Executable when initiated:** No
- **Related Commands:** *LRN?, *RST, *SAV, VINStrument:CONFigure:LBUS:RESet
- ***RST Condition:** all saved states set to the same state as the *RST state

*RMC

RMC** <name***> purges a single macro definition.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
name	string	1 through 12 characters	none

Comments

- Use the *PMC command to purge all macro definitions in one command.
- **Executable when initiated:** Yes
- **Related commands:** *DMC, *PMC
- ***RST Condition:** none

*RST

***RST** resets the Agilent E1488A as follows:

- Sets all commands to their *RST state.
- Aborts any data transfer in progress.
- Sets all stored (*SAV) states to the same state as the *RST state.

*RST does not affect:

- The state of VXIbus word serial protocol
- The output queue
- The Service Request Enable Register
- The Standard Event Status Enable Register
- The enable masks for the OPERation Status and Questionable Signal registers
- Memory data

Note After sending *RST, the VINStrument:LBUS:RESet command (or equivalent, depending on module type) *must* be sent all Local Bus modules to the right of the Agilent E1488A, in left-to-right order. This is necessary to ensure proper start-up of the *next* Local Bus transfer.

Comments

- **Executable when initiated:** Yes
- **Related Commands:** VINStrument:CONFigure:LBUS:RESet
- ***RST Condition:** none

*SAV

***SAV** **<*number*>** stores the current programming state into one of the 10 possible stored state areas. *Number* indicates which of the stored state areas should be used.

This command stores the states of all commands affected by *RST. Notable exceptions include memory contents (TRACe|DATA:DATA) and the STATus subsystem.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
number	numeric	0 through 9	none

Comments

- **Executable when initiated:** No
- **Related Commands:** *LRN?, *RCL, *RST

- ***RST Condition:** all saved states set to the same state as the *RST state

*SRE and *SRE?

***SRE** **<*mask*>** specifies which bits of the Status Byte Register are enabled to generate a IEEE-488.1 service request. Event and summary bits are always set and cleared in the Status Byte Register regardless of the enable mask. *Mask* is the sum of the decimal weights of the bits to be enabled.

The query form returns the current enable mask.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
mask	numeric	0 through 255	none

A 1 in a bit position enables service request generation when the corresponding Status Byte Register bit is set; a 0 disables it.

Comments

- **Executable when initiated:** Yes
- ***RST Condition:** unaffected
- **Power-On Condition:** no bits are enabled

Example

Enable service request on Message Available bit

***SRE 16** *Enable request on MAV*

*STB?

***STB?** returns the value of the Status Byte Register. Bit 6 (decimal weight 64) is set if a service request is pending.

Comments

- **Executable when initiated:** Yes
- **Related commands:** *SRE
- ***RST Condition:** none

*TST?

***TST?** causes the Agilent E1488A to execute its internal self-test and return a value indicating the results of the test.

A zero (0) response indicates that the self-test passed. A one (1) response indicates that the test failed. The failure also generates an error message with additional information on why the test failed.

Caution **The test destroys any data in memory (TRACe|DATA:DATA).**

Comments

- To minimize the likelihood of data loss, the *TST command may only be executed when no trace name is defined (TRACe|DATA:DEFine command). Use the TRACe|DATA:DELete command to delete an existing trace name before executing *TST.
- **Executable when initiated:** No
- ***RST Condition:** none

*WAI

***WAI** causes the Agilent E1488A to wait for all pending operations to complete before executing any further commands.

If STATus:OPC:INITiate OFF is set, command execution resumes when all commands received prior to the *WAI have been executed. If ON is set, *WAI waits for data transfers to complete before resuming command execution.

Comments

- **Executable when initiated:** Yes
- **Related commands:** *OPC, *OPC?
- ***RST Condition:** none

Agilent E1488A Instrument Memory Module Command Quick Reference

The following tables summarize the SCPI commands and IEEE 488.2 Common (*) commands for the Agilent E1488A Instrument Memory Module (i.e., Memory Module).

IEEE 488.2 Common Commands Quick Reference

Command	Title	Description
*CLS	Clear Status Command	Clears the Standard Event Status Register, the Operation Status Register, the Questionable Register, and the error queue.
*DMC	Define Macro Command	*DMC <*name*>,<*data*> creates a macro with the specified name and assigns zero, one, or a sequence of commands to the name.
*EMC	Enable Macro Command	*EMC <*enable*> enables (<*enable*> = -32768 to 32767) and disables (<*enable*> = 0) macro usage.
*EMC	Enable Macro Query	Queries if macro usage is enabled or disabled.
*ESE	Standard Event Status Enable Command	*ESE <mask> enables one or more event bits of the Standard Event Status Register to be reported in bit 5 (the Standard Event Status Summary Bit) of the Status Byte Register.
*ESE?	Standard Event Status Enable Query	Returns the current Standard Event Status Enable mask byte.
*ESR?	Standard Event Status Register Query	Returns the value of the Standard Event Status Register.
*IDN?	Identification Query	Returns the instrument ID as four groups separated by commas e.g. HEWLETT-PACKARD,E1488,0,A.01.00
*LMC?	Learn Macro Query	Returns list of macro names.
*LRN?	Learn Device Setup Query	Returns a sequence of commands that may be resent to the Agilent E1488A to return it to its current programming state.
*OPC	Operation Complete Command	Causes the Agilent E1488A to wait for all pending operations to complete (sets the Operation Complete Bit, bit 0, in the Standard Event Status Register).
*OPC?	Operation Complete Query	Returns a “1" when all pending operations have finished.
*PMC	Purge Macros Commands	Purges all macros.
*RMC	Remove Individual Macro Command	*RMC <*name*> purges a single macro definition.

Command	Title	Description
*RCL	Recall Command	*RCL <*number*> restores a previously stored programming state from one of the 10 possible stored state areas.
*RST	Reset	• *RST Conditions: FORMat[:DATA] ASCii VINStrument[:CONFigure]:LBUS[:MODE] OFF VINStrument[:CONFigure]:VME[:MODE] OFF places Agilent E1488 into the idle state clears all bits of both condition registers
*SAV	Save Command	*SAV <*number*> stores the current programming state into one of the 10 possible stored state areas.
*SRE	Service Request Enable Command	*SRE <*mask*> specifies which bits of the Status Byte Register are enabled to generate a IEEE-488.1 service request (<*mask*> is the sum of the decimal weights of the bits to be enabled).
*SRE?	Service Request Enable Query	Returns the current Service Request Enable mask.
*STB?	Status Byte Register Query	Returns the current value of the Status Byte Register.
*TST?	Test Query	Performs a self-test and responds with the test result. A 0 response indicates the test passed. A non-zero response indicates the test failed.
*WAI	Wait To Continue Command	Causes the Agilent E1488A to wait for all pending operations to complete before executing any further commands.

SCPI Commands Quick Reference

In the following quick reference, "co" means command only (i.e., command has no corresponding query command) and "qo" means query only (command is a query command). No designator means the command has a corresponding query command.

ABORt	co	Places Local Bus in the transfer idle state.
DISPlay :MONitor [:STATe] OFF | ON | 0 | 1		Disables or enables monitor mode for external display.
FORMat [:DATA] <*mode*>[,<*length*>]		Specifies returned data format using TRACe | DATA:DATA? query.
INITiate :CONTinuous OFF | ON | 0 | 1 [:IMMediate]	 co	Starts a Local Bus data transfer after any pauses. Starts a Local bus data transfer.
MEMory :FREE? :NONVolatile?	 qo qo	 Returns available memory and usage Returns 1 if backup switch is enabled.
STATus :OPC :INITiate OFF | ON | 0 | 1 :OPERation | QUEStionable :CONDition? :ENABle <*mask*> [:EVENt]? :NTRansition <*mask*> :PTRansition <*mask*> :PRESet	 qo qo co	 Controls wether the *OPC, *OPC?, and *WAI will complete immediately. Returns the contents of the condition register. Specifies which bits of the associated event register are included in its summary bit. Returns the contents of the appropriate event register. Sets the negative transition mask. Sets the positive transition mask. Initializes the enable registers and transition masks for the Operational and Questionable Signal status registers.
SYSTem :ERRor? :VERSion?	 qo qo	 Returns the error messages in the error queue. Returns the SCPI version number.
TRACE | DATA :CATalog? [:DATA] <*name*>,<*data_list*> :COUNt <*name*>,<*length*> :PARTial? <*name*>,<*start*>,<*stop*> :DATA? <name> :DEFine <*name*>,<*length*> :DELete :ALL [:NAME] <*name*>	 qo co qo qo co co	 Returns name of currently defined trace Stores data into the module. Overrides amount of data stored. Returns a portion of data stored. Returns data from memory. Creates a trace. Deletes any defined trace. Deletes specified trace.

VINStrument		
[:CONFigure]		
:LBUS		
[:MODE] <*mode*>		Selects operating mode for Local Bus.
:PAUSed?	qo	Returns 1 if Local Bus circuitry is in paused or 0 if LocalBus transfer is in progress.
:RESet	co	Resets the Local Bus circuitry.
:TEST		
:DATA <*block*>	co	Configures module for Local Bus testing.
:DATA? <*length*>	qo	Configures module to receive data for Local Bus testing, initiates the module, and returns the data.
:VME		
[:MODE] <*mode*>		Prepares module for register-level VME data transfer.
:RECeive		
:ADDRess		
:DATA?	qo	Returns A16,32 values.
:READy?	qo	Returns A16,38 values.
:SEND		
:ADDRess		
:DATA?	qo	Returns A16,32 values.
:READy?	qo	Returns A16,36 values.
:IDENtity?	qo	Returns module's vertual instrument capability.

SCPI Conformance Information

The Agilent E1488A Instrument Memory Module (Memory Module) conforms to the SCPI-1990.0 standard.

The following tables list all the SCPI confirmed and non-SCPI commands that the module can execute.

SCPI-Confirmed Commands

```
ABORt

:FORMat
   [:DATA]  <format>[,<length>]

INITiate
   [:IMMediate]
   :CONTinuous  <state>

MEMory
   :FREE?

STATus
   :PRESet
   :QUEStionable|:OPERation
      :CONDition?
      :ENABle  <mask>
      [:EVENt]?
      :NTRansition  <mask>
      :PTRansition  <mask>

SYSTem
   :ERRor?
   :VERSion?

TRACe|DATA
   :CATalog?
   [:DATA]  <name>,<data_list>
   :DEFine  <name>,<length>
   :DELete
      :ALL
      [:NAME]  <name>
```

Non-SCPI Commands

```
DISPlay
   :MONitor
      [:STATe]  <state>

MEMory
:NONVolatile?

STATus
   :OPC
      :INITiate  <state>

TRACe|DATA
   [:DATA]
      :COUNt  <name>,<length>
      :PARTial?  <name>,<start>,<stop>

VINStrument
   :IDENtity?
   [:CONFigure]
      :LBUS
         [:MODE]  <mode>
         :PAUSed?
         :RESet
      :TEST
         :DATA  <block>
         :DATA?  <length>
      :VME
         [:MODE]  <mode>
         :RECeive
            :ADDRess
               :DATA?
               :READy?
         :SEND
            :ADDRess
               :DATA?
               :READy?
```

Notes

Appendix A
Specifications

Contents

This appendix contains the Agilent E1488A Instrument Memory operating specifications. Except as noted, the specifications apply under the following conditions:

- **Period:** 1 year
- **Temperature:** 0° - 55° C
- **Relative humidity:** ≤ 65% @ 0° - 40° C
- **Warm up time:** 1 hour

"Typical", "typ", or "nominal" values are non-warranted supplementary information provided for applications assistance.

Instrument Specifications

Memory size:	2 to 16 MBytes
Memory type:	Low power static RAM
Data transfer speed:	**VXI Local Bus:** greater than 80 MBytes/s **VXI Data Transfer Bus:** greater than 2.5 MBytes/s in 16-bit mode or 5 MBytes/s in 32-bit mode nominal
Local Bus mode switch time:	2 nS switching from generating or consuming data to passing data through from LBUSA to LBUSC nominal
Compatible products:	Agilent E1429B Digitizer, Agilent E1430A Digitizer, Agilent E1445A Arbitrary Function Generator, Agilent E1485A Digital Signal Processor
Backup (nonvolatility):	On-board battery, VXI "+5VSTBY", or faceplate BNC **Caution: The maximum voltage the can be applied for backup is +4.5Vdc. Exceeding this voltage will damage the module.**
Battery life:	Nominally 2 months at 25° C, 7 days at 70° C with maximimum amount of SRAM installed
Faceplate BNC:	+3.5Vdc to +4.5 Vdc (see Caution above) connection for external backup power source. 5 mA maximum current

Interrupts: Upon reaching end of programmed memory transfer

VXI Local Bus usage: high-speed data transfers; consume, generate, or pass through data from LBUSA to LBUSC

General VXI Characteristics

Size: C

Slots: 1

Connectors: P1, P2

Weight (kg): 1.2 to 1.7, depending on memory size

Device Type: Register-based

VXIBus Revision Compliance: 1.4

Register Level Documentation: Yes

SCPI Revision: 1993.0

Manufacturer Code: 4095 Decimal

Model Code: 455 Decimal

Slave: A16 D16/limited D32
(Send/Receive Data registers only)

Power Supplies:

		Current (Amps)
+5v:	I(pm)	3.8
	I(dm)	0.01
-5.2v:	I(pm)	1.9
	I(dm)	0.01
-2v:	I(pm)	0.8
	I(dm)	0.01
+5vs:	I(pm)	0.005
	I(dm)	0.001

Average Watts/Slot: 27.5

ΔPressure(mm H_2O): 0.25

AirFlow (liters/s): 2.6

Appendix B
Error Codes

This chapter lists all error codes generated by the Agilent E1488A Instrument Memory Module (Memory Module).

Code	Message	Description
-101	Invalid character	Unrecognized character in parameter.
-102	Syntax error	Command is missing a space or comma between parameters.
-103	Invalid separator	Parameter is separated by a character other than a comma.
-104	Data type error	The wrong data type (number, character, string, expression) was used when specifying the parameter.
-108	Parameter not allowed	Parameter specified in a command which does not require one.
-109	Missing parameter	Command requires a parameter(s).
-112	Program mnemonic too long	Command keyword > 12 characters
-113	Undefined header	Command header (keyword) was incorrectly specified.
-121	Invalid character in number	A character other than a period or number is in the middle of a number.
-123	Numeric overflow	A parameter value is greater than what can be represented with the number format.
-124	Too many digits	More than 256 digits were used to specify a number.
-128	Numeric data not allowed	A number was specified when a letter was required.
-138	Suffix not allowed	Parameter suffix is specified when one is not allowed.
-141	Invalid character data	The wrong parameter type (discrete, boolean, numeric) was specified.
-144	Character data too long	Trace name is too long, or a discrete parameter is > 12 characters. Trace names must be 12 characters or less.
-148	Character data not allowed	Character data was specified when another parameter type (i.e. discrete, numeric, boolean) is required.
-158	String data not allowed	A string was specified when another parameter type (i.e. discrete, numeric, boolean) is required.
-161	Invalid block data	The number of bytes in a definite length data block does not equal the number of bytes indicated by the block header.

Code	Message	Description
-168	Block data not allowed	Block data was specified when another parameter type (i.e. discrete, numeric, boolean) is required.
-178	Expression data not allowed	The parameter was specified as an expression.
-183	Invalid inside macro definition	Trace data list is inside a macro.
-213	Init ignored	INITiate:IMMediate received while the E1488A was currently initiated, or INITiate:CONTinuous ON received while VINstrument:LBUS:MODE is CONSume|EAVesdrop|GENerate|MGENerate.
-221	Settings conflict	*OPC, *OPC?, or *WAI received when STATus:OPC:INItiate ON and INITitate:CONTinuous ON set. Command would deadlock.
-272	Macro execution error	Macro program data sequence could not be executed due to a syntax error within the macro definition.
-273	Illegal macro label	The macro label defined in the *DMC command was too long, the same as a common command keyword, or contained invalid header syntax.
-276	Macro recursion error	A macro program data sequence could not be executed because the sequence leads to the execution of a macro being defined.
-277	Macro redefinition not allowed	A macro label in the *DMC command could not be executed because the macro label was already defined.
-330	Self-test failed	Note the information associated with the message for a description of the failure.
-350	Too many errors	The E1488A error queue is full and additional errors have occurred.
-410	Query INTERRUPTED	The E1488A was sent a command before it was finished responding to a query command.
-420	Query UNTERMINATED	The controller (computer) attempted to read a query response from the E1488A without having first sent a complete query command.
-430	Query DEADLOCKED	The E1488A's input and output buffers are full and the E1488A cannot continue.
1000	Out of memory	TRACe:DEFine size is greater than installed memory size.
1001	Trace name not defined	Command requires previous definition of a trace name, or specified trace name does not exist.
1002	Trace already defined	Trying to redefine an existing trace name or define a new trace name while another already exists.
1003	No trace data stored	Trying to read or send trace data when none has been stored.
1004	Insufficient trace data stored	Trying to send trace data when no data is stored.

Code	Message	Description
1005	Trace defined size too small	Trying to receive Local Bus data when defined trace size is less than 16 bytes.
1011	Illegal while initiated	Command cannot be executed while the E1488A is in the initiated state.
1012	Illegal while VINS:CONF:VME:MODE not OFF	Command cannot be executed while VME register access enabled.
1013	Illegal while overlap mode OFF set	Trying to enable Initiated bit in Operation status register while cscpi_overlap(0) set (Compiled SCPI only).
1021	Local bus test data size not multiple of 4 bytes	VINStrument:TEST:DATA command/query data size must be multiple of 4 bytes.
1022	Not local bus test data received	VINStrument:TEST:DATA? received no data.
1023	Incorrect amount of local bus test data received	VINStrument:TEST:DATA? received more or less data than specified.

Appendix C
Register-Based Programming

Contents

This appendix gives information on the Agilent E1488A Instrument Memory Module's registers and a register-based programming example. The sections in this appendix are:

- General Information 131
- Register Addressing 131
  - The Base Address 133
- Memory Module Registers 134
  - Register Assignments 134
  - Register Descriptions 135
- Register-Based Example Program 148
  - System Configuration 148
  - Running the Program 148
  - Program Example 148

General Information

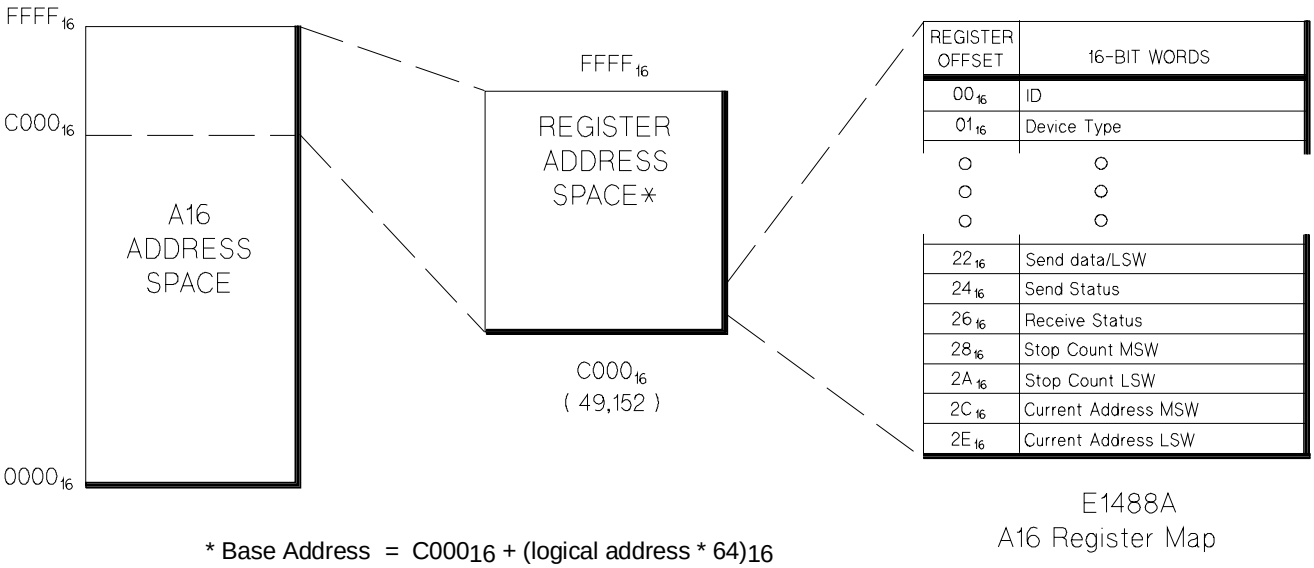
The Agilent E1488A Instrument Memory Module (Memory Module) is a register-based device which does not support the VXIbus word serial protocol. When sending a SCPI command to the module, the Memory Module driver in the Agilent E1405 or Agilent E1406 Command Module interprets the command and writes the information to the module's registers.

Register Programming is a series of writes **directly** to the module's registers. When using an embedded controller, it increases throughput speed since it eliminates command interpreting and the controller accesses the registers directly from the backplane.

Register Addressing

Register addressing for register-based modules are located in the upper 25% of the VXI A16 address space. Allocated to every VXI device (up to 256 devices) is a 64 byte block of addresses.

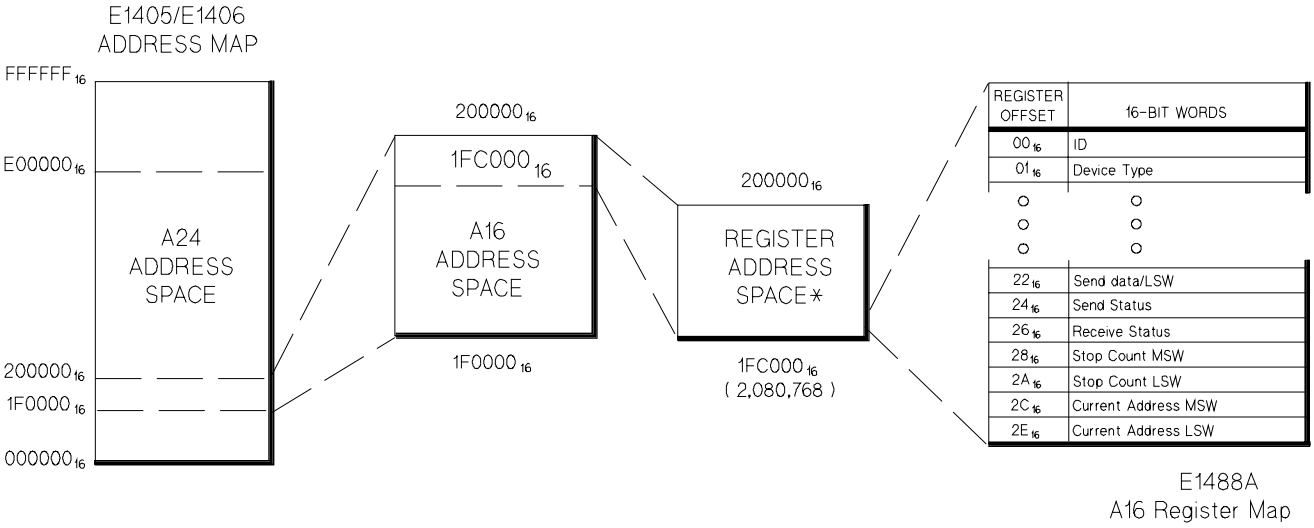
Figure C-1 shows the register address location within A16. Figure C-2 shows the location of A16 address space in the Agilent E1405 or E1406 Command Module.



* Base Address = $C000_{16}$ + (logical address * 64)$_{16}$
= 49,152 + (logical address * 64)$_{10}$

Register Address = Base Address + Register Offsett (i.e., Register Number)

Figure C-1. Memory Module Registers within A16 Space



* Base Address = $1FC000_{16}$ + (logical address * 64)$_{16}$
= 2,080,768 + (logical address * 64)$_{10}$

Register Address = Base Address + Register Offsett (i.e., Register Number)

Figure C-2. Memory Module Registers within Command Module A16 Space

The Base Address

When writing or reading to/from a register, specify a hexadecimal or decimal register address. This address consists of an A16 base address plus a register offset or register number. The A16 base address used in register-based programming depends on whether the A16 address space is located in the Agilent E1405/E1406 Command Module or an embedded controller (like the Agilent V/382). Use Figures C-1 and C-2, and Table C-1 to determine the base address for the following configurations.

- Agilent Instrument BASIC (IBASIC) in the Agilent E1405/E1406 Command Module
- External Controller over GPIB to an Agilent E1405/E1406 Command Module
- Embedded Controller

Note If using an embedded controller, like the Agilent V/382, with the Standard Instrument Control Library (SICL), you do not need to determine the base address. SICL only needs the module's Logical Address to program the module.

Table C-1. Controller Configurations and Base Addresses

Control Method	Programming Method	Base Address
Instrument BASIC (IBASIC) (Absolute Programming)	READIO -9826,Base_addr + offset WRITEIO -9826,Base_addr + offset;data	Base_addr = $1FC00_{16}$ + (LADDR * 64)$_{16}$ = 2,080,768 + (LADDR * 64) offset = register offset (see Figure C-1)
(Using Select Code 8)	READIO 8,Base_addr + reg_number WRITEIO 8,Base_addr + reg_number;data	Base_addr = LADDR * 256 reg_number = (register offset) / 2
External Controller (connected to Agilent E1405/E1406 Command Module over GPIB)	VXI:READ? LADDR,offset VXI:WRITE LADDR,offset,data DIAG:PEEK? Base_addr + offset,width DIAG:POKE Base_addr + offset,width,data	offset = register offset (see Figure C-1) Base_addr = $1FC00_{16}$ + (LADDR * 64)$_{16}$ = 2,080,768 + (LADDR * 64) offset = register offset (see Figure C-1) width = 8 or 16 bytes (use 16 bytes)
Embedded Controller	Method depends on Controller used (see Controller documentation)	Base_addr = $C00_{16}$ + (LADDR * 64)$_{16}$ = 49,152 + (LADDR * 64) offset = register offset (see Figure C-2)
LADDR = Memory Module's Logical Address		

Memory Module Registers

This sections explains the registers of the Memory Module.

Register Assignments

Register assignments for the Agilent E1488A Memory Module are as follows:

Register Number	Read	Write
00_{16}	ID	*unused*
02_{16}	Device Type	*unused*
04_{16}	VXI Status	VXI Control
06_{16}	IRQ control	IRQ control
08_{16}	Memory Attribute	*unused*
$0A_{16}$ through 10_{16}	*unused*	*unused*
12_{16}	Local Bus Reset	Local Bus Reset
14_{16}	Local Bus Restart	Local Bus Restart
16_{16}	Local Bus Flush	Local Bus Flush
18_{16}	Port Control	Port Control
$1A_{16}$	Common Capabilities	*unused*
$1C_{16}$	Description	*unused*
$1E_{16}$	Subclass	*unused*
20_{16}	Send Data MSW	Receive Data MSW
22_{16}	Send Data/LSW	Receive Data/LSW
24_{16}	Send Status	*unused*
26_{16}	Receive Status	*unused*
28_{16}	Stop Count MSW	Stop Count MSW
$2A_{16}$	Stop Count LSW	Stop Count LSW
$2C_{16}$	Current Address MSW	*unused*
$2E_{16}$	Current Address LSW	*unused*

Note

Access the Send Data and Receive Data registers via either VME D16 (16-bit) or D32 (32-bit) transfers. When accessing these registers via D16 transfers, first access the MSW register (offset 20h), then the LSW register.

Access all other registers only by the VME D16 (16-bit) access. The Agilent E1488A responds to D32 transfers to these registers as if a D16 transfer were being don, i.e., only one register will be read/written to, not two.

Reads of unused registers will return all 1's ($FFFF_{16}$). Writes to unused registers have no effect.

Register Descriptions

The following describes the Memory Module registers.

ID Register

The ID read register indicates the classification, addressing mode, and the manufacturer of the Agilent E1488A. The values below indicate that the Agilent E1488A is an extended, A16-only module manufactured by Agilent Technologies.

Address 00_{16}	15	14	13	12	11 - 0
Description	Device Class		Address Space		Manufacturer ID
Value	01_2		11_2		FFF_{16}

Device Type Register

The module's Device Type read register indicates A24/A32 memory requirements (none) and the model code for the Agilent E1488A.

Address 02_{16}	15	14	13	12	11 - 0
Description	A24/32 Requirement				Model Code
Value	0000_2				$1C7_{16}$

VXI Status Register

The VXI Status read register indicates the status of the Agilent E1488A.

Address 04_{16}	15	14	13 - 2	1	0
Description	*unused*	MODID*	*unused*	DONE	PAUSED

Unused: not used by the E1488A. These bits will be all 1's when read.

MODID*: A one (1) in this bit indicates that the module is not selected via the P2 MODID line. A zero (0) indicates that the module is selected by a high state on the P2 MODID line.

DONE: A one (1) in this bit indicates that the Agilent E1488A has reached the transfer count specified by the Stop Count registers. A zero (0) indicates the transfer count has not been reached. This bit is cleared by a write to the Local Bus Reset register or by setting both the Send Port and Receive Port fields of the Port Control register to 0's.

PAUSED: A one (1) in this bit indicates that the Local Bus Controller chip is is the Paused state. The Local Bus Controller chip may be reconfigured and the memory may be accessed via the VXIbus. A write to the Local Bus Restart register must be done to restart Local Bus I/O. A zero (0) indicates that the Local Bus I/O is in progress and should not be disturbed.

VXI Control Register

The VXI Control write register is used to reset the module and to disable driving of the SYSFAIL line.

Address 04_{16}	15 - 2	1	0
Description	*unused*	SYSINH	RESET

Unused: The value of these bits are ignored.

SYSINH: Writing a one (1) to this bit prevents the module from asserting the SYSFAIL line. Writing a zero (0) permits the module to drive SYSFAIL during reset.

RESET: Writing a one (1) resets the module. Writing a zero (0) causes the Agilent E1488A to restart and return to its power-on configuration.

IRQ Control Register

The IRQ Control read/write register is used to control interrupt generation of the Agilent E1488A.

Address 04_{16}	15 - 2	1	0
Description	*unused*	Done IRQ En	Paused IRQ En

Unused: The value of these bits are ignored.

Done IRQ Enable: When set (1), enables interrupt generation when the transfer count specified by the Stop Count Registers is reached. When clear

(0), disables interrupt generation when a transfer count is reached. Clearing the enable bit does not clear a pending interrupt.

Paused IRQ Enable: When set (1), enables interrupt generation when the Local Bus circuitry enters the Paused (inactive) state. When clear (0), disables interrupt generation on Paused. Clearing the enable bit does not clear a pending interrupt.

Note Interrupt generation on Paused cannot be enabled when the LB Continuous bit of the Port Control register is set (1). In continuous mode, the Local Bus circuitry briefly (about 2 nS) transitions into and out of the Paused state when the Frame indication is received. This brief transition generates a signal too short to reliably generate an interrupt. The Agilent E1488A therefore forces the Paused interrupt bit to be clear (0) while the LB Continuous bit is set.

Memory Attribute Register

The Memory Attribute read register indicates important memory characteristics of the Agilent E1488A.

Address 08_{16}	15-14	13	12	11	10-8
Description	Memory Type	Access Modes	Block Transfer	Non-Volatile	Access Speed
Value	11_2	1	1	*see below*	100_2

Address 08_{16}	7	6-4	3	2-0
Description	D32	*Reserved*	SIMM Size	Blocks Loaded
Value	0	111_2	*see below*	*see below*

Memory Type: the value indicates that the Agilent E1488A contains RAM.

Access Modes: the value indicates that the memory may be accessed in both Non-Priviledged and Supervisory modes.

Block Transfer: the value indicates that the Agilent E1488A does *not* have Block Transfer Capability.

Non-volatile: a zero (0) in this bit indicates that the Backup Enable DIP switch has been set to the ENABLED position and that the memory is non-volatile (assuming a backup power source is available). A one (1) indicates that the DIP switch is in the DISABLED position and that the memory is volatile.

Access Speed: the value indicates that the Agilent E1488A has an access speed that is between 400 nS and 800 nS.

D32: the value indicates that the Agilent E1488A supports both D16 and D32 access to the memory.

Reserved: these bits are reserved for future use by the VXI standard.

SIMM Size: a zero (0) in this bit indicates that 512 KByte SIMMs are loaded, as specified by the SIMM Size DIP switch. A one (1) indicates that 2 MByte SIMMs are loaded. (2 Mbyte SIMMs are not available at the time this manual was published.)

Blocks Loaded: indicates the number of the last SIMM block loaded, as specified by the SIMM Blocks Loaded DIP switch. A value of 000_2 indicates that only block 0 is loaded, 001_2 that blocks 0 and 1 are loaded, etc. Since each block uses 4 SIMMs, the total memory capacity is given by:

Capacity = 4 * (SIMM Size) * (Blocks Loaded + 1)

Note The SIMM Size and Blocks Loaded bits simply reflect the respective DIP switch settings. If the switch settings are incorrect, the memory capacity indicated by the SIMM Size and Blocks Loaded bits will be incorrect.

Local Bus Reset Register

Any read or write to this register will reset the Local Bus Controller chip, halting any transfer in progress and placing the Local Bus in the Paused (inactive) state. The Local Bus Enable bit (bit 0) and the Local Bus Continuous bit (bit 12) of the Port Control register are cleared. No other register contents are affected.

Note After resetting the Local Bus Controller chip, reset the Local Bus Controller chips of all Local Bus modules to the right of the Agilent E1488A in a left-to-right order to ensure proper start-up of the next Local Bus transfer.

Local Bus Restart Register

A read or write to this register will initiate a new Local Bus transfer. The Local Bus Enable bit (bit 0) of the Port Control register must have been previously set (1). Since setting the Local Bus Enable bit initiates a Local Bus Transfer, this register is most useful in quickly starting a new transfer after a previous one has finished (Local Bus in the Paused state).

Local Bus Flush Register

In receiving data from the Local Bus, the Local Bus circuitry on the Agilent E1488A collects incoming data into groups of four bytes before sending it to the memory. At the end of a transfer that is not a multiple of four bytes, the extra bytes will not have been sent to the memory, but remain in the Local Bus Controller chip. Any read or write to the Local Bus Flush register will force any extra bytes to be sent to the memory and places the Local Bus circuitry in the Paused (inactive) state. If the transfer was a multiple of four bytes, the access will have no effect.

Note The Local Bus Pipe bit (bit 1) and Local Bus Continuous bit (bit 12) of the Port Control register should be set to 0 before the access to the Local Bus Flush register; otherwise, the extra bytes will be flushed and a new transfer cycle will be initiated.

Note This register should not be accessed while data transfers are in progress. Such an access will prematurely terminate a transfer. You should verify that the data generator has halted (the Paused bit is set (1) in the VXI Status register) before flushing out the extra bytes.

Port Control Register

The Port Control register controls the operation of the Local Bus circuitry and access to the memory from both the Local Bus and the VXIbus.

Address 18_{16}	15	14	13	12	11-8
Description	Block Inhibit	Frame Inhibit	LB Strip	LB Cont.	LB Mode

Address 18_{16}	7-5	4-2	1	0
Description	Send Port	Receive Port	LB Pipe	LB Enable

Block Inhibit: When set (1), inhibits the generation of the Local Bus Block bit on the last byte of data. This is useful when data from multiple Agilent

E1488A modules must be sent together as a single block of data. When clear (0), the Block bit is generated on the last byte of data.

Frame Inhibit: When set (1), inhibits the generation of the Local Bus Frame bit on the last byte of data. This is useful in the GP mode where the Agilent E1488A generates a block of data, then changes to pipeline mode to allow other modules to add their data to the frame. When clear (0), the Frame bit is generated on the last byte of data.

LB Strip: When set (1), clears the Local Bus Frame bit on any data being piped through the Agilent E1488A. When clear (0), the Frame bit is piped unchanged.

LB Continuous: When set (1), enables continuous mode on the Local Bus. In this mode, the Local Bus circuitry automatically restarts itself immediately and again whenever it transitions into the Paused state. The current values of the LB Mode and LB Strip bits are used when the circuitry restarts.

When clear (0), continuous mode is disabled. The Local Bus circuitry transition into the Paused state when the condition specified by the Local Bus Mode bits is reached. An access to the Local Bus Restart register or a 0-to-1 transition of the Local Bus Enable bit must occur to restart Local Bus operation.

Note The LB Enable bit should be kept cleared (0) whenever the LB Continuous bit is set (1). The LB Continuous bit should be kept cleared (0) whenever the LB Enable bit is set (1).

LB Mode: Specifies the mode of operation for the Local Bus, according to the following table:

Mode Value	Transfer Mode	Sequence	Description
0000_2	Pipe	PP...	Pipe, pause on End of Block
0001_2	Pipe	PP...	Pipe, pause on Frame
0010_2	Consume	CC...	Consume, pause on Done
0011_2	Eavesdrop	EE...	Eavesdrop, pause on Done
0100_2	Generate	GG...	Generate, pause on Done
0101_2	*illegal*		
0110_2	PC	PCPC...	Pipe, pause on Frame Consume, pause on Done
0111_2	PE	PEPE...	Pipe, pause/auto-restart on Frame Eavesdrop, pause on Done
1000_2	PG	PGPG...	Pipe, pause/auto-restart on Frame (strip Frame) Generate, pause on Done
1001_2	*illegal*		
1010_2	*illegal*		
1011_2	*illegal*		
1100_2	CP	CPCP...	Consume, pause/auto-restart on Done Pipe, pause on Frame
1101_2	EP	EPEP...	Eavesdrop, pause/auto-restart on Done Pipe, pause on Frame
1110_2	GP	GPGP...	Generate, pause/auto-restart on Done Pipe,pause on Frame
1111_2	*illegal*		

Setting the LB Mode field to an illegal value has undefined effects.

Setting the LB Pipe bit to 1 overrides the LB Mode field, and forces mode 0001_2 (Pipe, pause on Frame) to be enabled, regardless of the value programmed in the LB Mode field.

Pause/auto-restart indicates that the Local Bus Controller chip will pause on the specified condition. If the LB Continuous bit is set (1), the Local Bus Controller chip will automatically restart itself within about 2 nS after *every* pause. If the LB Continuous bit is clear (0), the Agilent E1488A will automatically restart the Local Bus Controller chip in the new mode after the first pause (*only*) after enabling a Local Bus operation (0-to-1 transition LB Enable bit). This auto-restart takes approximately 250 nS to complete. Subsequent pauses will only be restarted via an access to the Local Bus Restart register or by clearing and setting the LB Enable bit.

Send Port: specifies to which data port (Local Bus or VXIbus) memory data is to be sent, according to the following table:

Send Port Value	Destination
000_2	NONE
001_2	*illegal*
010_2	Local Bus
011_2	VXIbus
100_2	*illegal*
101_2	*illegal*
110_2	*illegal*
111_2	*illegal*

Attempting to set any illegal value will cause the Send Port field to be set to NONE (000_2). Since the Agilent E1488A does not support simultaneous read/write of the memory, only one of the Send Port and Receive Port fields may be set to a value other than NONE. Attempting to set both fields to other values will cause the Send Port field to be set to NONE.

Note The Agilent E1488A uses a pipelined memory architecture. When changing the Send Port field from NONE (000_2) to either VXIbus or Local Bus, the pipeline must be filled with valid data by four 32-bit accesses to the Send Data MSW register or four 16-bit accesses to the Send Data LSW register. Failure to do this will cause 16 bytes of indeterminate data to be sent before any valid data.

Receive Port: specifies from which data port (Local Bus or VXIbus) memory data is to be received, according to the following table:

Send Port Value	Destination
000_2	NONE
001_2	*illegal*
010_2	Local Bus
011_2	VXIbus
100_2	*illegal*
101_2	*illegal*
110_2	*illegal*
111_2	*illegal*

Attempting to set any illegal value will cause the Receive Port field to be set to NONE (000_2). Since the Agilent E1488A does not support simultaneous read/write of the memory, only one of the Send Port and Receive Port fields may be set to a value other than NONE. Attempting to set both fields to other values will cause the Send Port field to be set to NONE.

Note The Agilent E1488A uses a pipelined memory architecture. After transferring data to the Agilent E1488A from either the VXIbus or Local Bus, and before changing the Receive Port field from either VXIbus or Local Bus to NONE (000_2), the pipeline must be flushed by a 32-bit write of any data to the Receive Data MSW register or a 16-bit write of any data to the Receive Data LSW register. Failure to do this will cause up to the last 4 bytes of data to be lost.

LB Pipe: When set (1), the Local Bus circuitry is forced into pipeline mode. The value of the LB Mode bits is forced to 0001_2 (Pipeline, pause on frame). Pipelining of data starts following a subsequent 0-to-1 transition of the LB Enable bit or of the LB Continuous bit. When clear (0), the Local Bus operating mode is controlled by the LB Mode bits.

LB Enable: The 0-to-1 transition of this bit initiates a Local Bus transfer cycle. A 1-to-0 transition while the Local Bus circuitry is in the active state will reset the Local Bus circuitry (exactly as if the Local Bus Reset register had been written to), thus placing the Local Bus circuitry into the inactive state.

Note The LB Enable bit should be kept cleared (0) whenever the LB Continuous bit is set (1). The LB Continuous bit should be kept cleared (0) whenever the LB Enable bit is set (1).

Note After resetting the Local Bus Controller chip via a 1-to-0 transition of the LB Enable bit while the Local Bus circuitry was in the active state, the Local Bus Controller chips of all Local Bus modules of any model number to the right (next higher slot) of the Agilent E1488A should also be reset to insure proper starting of the next Local Bus transfer.

Common Capabilities Register

This register, defined by the Agilent Technologies Virtual Instrument/Local Bus Specification, indicates which of certain optional capabilities are implemented by the Agilent E1488A. A read returns all 0's, and indicates that none of these capabilities are implemented.

Description Register

This register indicates to which revision of the Agilent Technologies Virtual Instrument/Local Bus Specification the Agilent E1488A is designed, the module type, and what access modes are implemented. The values below indicate that Agilent E1488A was designed to Revision 1.0 of the specification, is an unspecified data storage module, that its virtual instrument registers are in A16 address space, and that VME Slave Send/Receive and Local Bus Send/Receive are implemented.

Address $1C_{16}$	15-14	13-11	10-8	7	6
Description	Revision	Subtype	Module Type	*unused*	A16
Value	11_2	111_2	111_2	0	1

Address $1C_{16}$	5	4	3	2	1	0
Description	VME Slave Send	VME Slave Receive	VME Master Send	VME Master Receive	LBUS Send	LBUS Receive
Value	1	1	0	0	1	1

Subclass Register

This register indicates the subclass of the module. The values below indicate that the Agilent E1488A is a virtual instrument module as defined by the Agilent Technologies Virtual Instrument/Local Bus Specification.

Address $1E_{16}$	15	14-12	11-0
Description	Format	Manufacturer Subclass	Manufacturer ID
Value	0	111_2	FFF_{16}

Send/Receive Data MSW/LSW Registers

These registers provide access to the memory from the VXIbus. The appropriate Send Port or Receive Port field of the Port Control register must have been previously set to VXIbus.

Memory data is always 32 bits wide. When accessing memory using D32 access, only the MSW (Most Significant Word) register should be read or written. When accessing memory using D16 accesses, the MSW (Most Significant Word) register should be read/written first, followed by the LSW register. When writing to memory, data is not actually written to the memory until the Receive Data LSW register is written.

Note

The memory address counter is incremented by a 32-bit access to the MSW register or a 16-bit access to the LSW register. Therefore, if the LSW register is accessed first, a subsequent access to the MSW register will read/write data for the next memory address.

Send Status Register

This read-only register contains status information for the pacing of data from the Send Data registers. For the Agilent E1488A, only the least significant bit of this register is important.

Address 24_{16}	15-11	10-9	8-7	6-2	1	0
Description	unused	Count Width	Data Width	Block Size	Block Ready	Data Ready
Value	0000_2	00_2	11_2	0000_2	0	*see below*

Count Width: The value indicates that Send Count register specified by the Agilent Technologies Virtual Instrument/Local Bus Specification is not implemented.

Data Width: The value indicates that the data size, as contained in the Send Data MSW/LSW registers is 32 bits wide.

Block Size/Ready: The values indicate that the Agilent E1488A does not support block data.

Data Ready: During a Agilent E1488A-to-VXIbus transfer (enabled when the Send Port bits of the Port Control register are set to VXIbus (010_2)), this bit is set (1); This indicates that data is available in the Send Data registers, until the current memory address reaches the address specified by the Stop Count registers. It is clear (0) otherwise.

Receive Status Register

This read-only register contains status information for the pacing of data from the Receive Data registers. For the Agilent E1488A, only the least significant bit of this register is important.

Address 24_{16}	15-11	10-9	8-7	6-2	1	0
Description	unused	Count Width	Data Width	Block Size	Block Ready	Data Ready
Value	0000_2	00_2	11_2	0000_2	0	*see below*

Count Width: The value indicates that Receive Count register specified by the Agilent Technologies Virtual Instrument/Local Bus Specification is not implemented.

Data Width: The value indicates that the data size, as contained in the Receive Data MSW/LSW registers is 32 bits wide.

Block Size/Ready: The values indicate that the Agilent E1488A does not support block data.

Data Ready: During a VXIbus-to-Agilent E1488A transfer (enabled when the Receive Port bits of the Port Control register are set to VXIbus (010_2)), this bit is set (1). This indicates that data is written to the Receive Data registers, until the current memory address reaches the address specified by the Stop Count registers. It is clear (0) otherwise.

Stop Count MSW/LSW Registers

This pair of 16-bit registers are used to specify the number of bytes to be involved in a Local Bus transfer, either send or receive. The most significant 16 bits are maintained in the Stop Count MSW register, the least significant 16 bits in the Stop Count LSW register.

Because of the pipelined architecture of the Agilent E1488A, the value that should be written to these registers varies depending on the direction (send or receive) and source or destination (VXIbus or Local Bus) of the transfer. To transfer *count* bytes, the value to write is given by the table below:

Transfer:	Local Bus	VXIbus
Send	*count* + 7	N/A
Receive	*count* - 13	N/A

Note that, for Local Bus receive transfers, because the value written to the Stop Count MSW/LSW registers must be a number greater than or equal to 0, no less than 13 bytes can be received for one transfer.

Current Address MSW/LSW Registers

This pair of 16-bit read-only registers provide access to the current memory address counter. The most significant 12 bits are maintained in the Current Address MSW register, the least significant 16 bits in the Current Address LSW register.

Note

The value of the memory address counter is dynamic while Local Bus transfers are in progress. The value of the counter is latched when the Current Address MSW register is read. Therefore, when reading these registers, the MSW register should be accessed first.

The counter is reset to all 0's when both the Send Port and Receive Port bits of the Port Control register are set to NONE (000_2). At the end of a receive transfer where the amount of data transferred may be unknown, these registers may be used to determine the amount of data received, but they should be read just before setting the Send Port and Receive Port bits to NONE (but after flushing any extra bytes from the Local Bus Controller chip and before flusing the memory pipeline). The number of bytes transferred is then the value read from these registers minus three.

Register-Based Example Program

The example program (register.c) in this section shows how to control three Memory Modules using their registers. The following identifies the system on which the prograam was written, how to compile the program, and lists the program.

System Configuration

The program was developed on the following system:

Controller: Agilent E1499A V/382 Controller

Operating System: HP-UX version 9.0

Mainframe: Agilent 75000 Series C

Agilent E1488A Logical Addresses: 200, 208, and 216

Running the Program

To run the program, first compile and link the program to make an executeable file. The program was compiled using the C compiler of HP-UX version 9.0 and linked with the *lsicl* library from the Standard Instrument Control Library (SICL). The program was compiled and linked as follows:

```
cc -Aa -o register register.c +OV -lsicl
```

Program Example

This program uses three Memory Modules to store and output data over the VMEbus. Assumed is a memory size of 2097152 bytes. The program operates as follows:

- setup first module to receive data from the VXIbus; then store the data
- setup first module to output data (generate)
- setup second module to pass data through (pipeline)
- setup third module to receive data (consume)
- initiate third module to receive data
- initiate second module to pass data
- initiate first module to output data
- data is now output by the first module, passed through the second module, and received and stored by the third module

The example program listing is as follows:

```
/* register.c - This program:                                    */
/*          1.) Stores 2097152 bytes into the first module       */
/*          2.) Passes data through the second module            */
/*          3.) Stores 2097152 bytes from the first module into the */
/*             third module                                      */
/*                                                               */
/* Assumed are the following Logical Addresses of the modules:   */
/*    Module 1: 200                                              */
/*    Module 2: 208                                              */
/*    Module 3: 216                                              */

#include  <stdio.h>
#include  <stdlib.h>
#include  <sicl.h>
#define  SICL_NULL ((INST)0)

/* Agilent E1488A register definitions */
/* Register offsets */

#define  control          0x04
#define  status           0x04
#define  mem_attrib       0x08
#define  lb_reset         0x12
#define  lb_restart       0x14
#define  lb_flush         0x16
#define  port_ctrl        0x18
#define  send_data        0x20
#define  rec_data         0x20
#define  send_data_lsw    0x22
#define  rec_data_lsw     0x22
#define  stop_cnt_msw     0x28
#define  stop_cnt_lsw     0x2A
#define  curr_addr_msw    0x2C
#define  curr_addr_lsw    0x2E

/* Control register definitions */
#define  CTRL_SYSFAIL_INH 0x0002
#define  CTRL_RESET       0x0001

/* Status register definitions */
#define  LB_PAUSED     0x0001

/* Memory Attribute register definitions */
#define  LOADED_MASK          0x0007
```

Continued on Next Page

```
/* Port Control register definitions */
#define  LB_IDLE                0x0000
#define  LB_ENABLE              0x0001
#define  LB_PIPE                0x0002
#define  REC_PORT_MASK          0x001C
#  define  REC_PORT_NONE        0x0000
#  define  REC_PORT_LBUS        0x0008
#  define  REC_PORT_VME         0x000C
#define  SEND_PORT_MASK         0x00E0
#  define  SEND_PORT_NONE       0x0000
#  define  SEND_PORT_LBUS       0x0040
#  define  SEND_PORT_VME        0x0060
#define  LB_MODE_SHIFT          8
#define  LB_MODE_MASK           0x0F00
#  define  MODE_PIPE            0x0100
#  define  MODE_CONSUME         0x0200
#  define  MODE_GENERATE        0x0400
#define  LB_CONTINUOUS          0x1000
#define  LB_STRIP               0x2000
#define  LB_FRAME_INH           0x4000
#define  LB_BLOCK_INH           0x8000

/* Stop Count register definitions */
#define  REC_OFFSET             (13L)
#define  SEND_OFFSET            (-7L)

main()
{
   INST E1488_1, E1488_2, E1488_3;
   char        *vme_reg_1, *vme_reg_2, *vme_reg_3;
   long        i, mem_amt, mem_loop, j;
   short       temp;
   static      const short ZERO = 0;

   /* open all three E1488A modules */
   E1488_1 = iopen("vxi,200");
   if (E1488_1 == SICL_NULL) {
       printf("Error opening Logical Address 200\n");
       exit(1);
   }
```

Continued on Next Page

```
vme_reg_1 = (char*)imap(E1488_1, I_MAP_VXIDEV, 0, 1, NULL);

E1488_2 = iopen("vxi,208");
if (E1488_2 == SICL_NULL) {
    printf("Error opening Logical Address 208\n");
    exit(1);
}
vme_reg_2 = (char*)imap(E1488_2, I_MAP_VXIDEV, 0, 1, NULL);

E1488_3 = iopen("vxi,216");
if (E1488_3 == SICL_NULL) {
    printf("Error opening Logical Address 216\n");
    exit(1);
}
vme_reg_3 = (char*)imap(E1488_3, I_MAP_VXIDEV, 0, 1, NULL);

/* Soft Reset modules; this it not necessary, but shows how to do it */
iwpoke((volatile short*)(vme_reg_1 + control), CTRL_SYSFAIL_INH);
iwpoke((volatile short*)(vme_reg_1 + control), CTRL_SYSFAIL_INH | CTRL_RESET);
iwpoke((volatile short*)(vme_reg_1 + control), CTRL_SYSFAIL_INH);

iwpoke((volatile short*)(vme_reg_2 + control), CTRL_SYSFAIL_INH);
iwpoke((volatile short*)(vme_reg_2 + control), CTRL_SYSFAIL_INH | CTRL_RESET);
iwpoke((volatile short*)(vme_reg_2 + control), CTRL_SYSFAIL_INH);

iwpoke((volatile short*)(vme_reg_3 + control), CTRL_SYSFAIL_INH);
iwpoke((volatile short*)(vme_reg_3 + control), CTRL_SYSFAIL_INH | CTRL_RESET);
iwpoke((volatile short*)(vme_reg_3 + control), CTRL_SYSFAIL_INH);

/* Determine memory size - assume left & right modules are same size */
temp = iwpeek((volatile short*)(vme_reg_1 + mem_attrib));
mem_amt = (long)((temp & LOADED_MASK) + 1) * 2097152L;
mem_loop = mem_amt / 4;

/* Set up left-hand module to store data from VME */
printf("Setting up left-hand module...\n");
iwpoke((volatile short*)(vme_reg_1 + port_ctrl), LB_IDLE);
iwpoke((volatile short*)(vme_reg_1 + port_ctrl), SEND_PORT_NONE | REC_PORT_VME);

/* Store data in left-hand module */
for (i = 1; i <= mem_loop;) {
    ilpoke((volatile long*)(vme_reg_1 + rec_data), i++);
    ilpoke((volatile long*)(vme_reg_1 + rec_data), i++);
    ilpoke((volatile long*)(vme_reg_1 + rec_data), i++);
    ilpoke((volatile long*)(vme_reg_1 + rec_data), i++);
}
```

Continued on Next Page

```
/* Flush last word to memory, set up for generate */
ilpoke((volatile long*)(vme_reg_1 + rec_data), ZERO);
iwpoke((volatile short*)(vme_reg_1 + port_ctrl), LB_IDLE);
iwpoke((volatile short*)(vme_reg_1 + port_ctrl),
      MODE_GENERATE | SEND_PORT_LBUS | REC_PORT_NONE);
iwpoke((volatile short*)(vme_reg_1 + lb_reset), ZERO);

/* Set up stop address */
i = mem_amt - SEND_OFFSET;
iwpoke((volatile short*)(vme_reg_1 + stop_cnt_msw), (short)(i >> 16));
iwpoke((volatile short*)(vme_reg_1 + stop_cnt_lsw), (short)i);

/* Set up memory read pipeline */
j = ilpeek((volatile long*)(vme_reg_1 + send_data));
j = ilpeek((volatile long*)(vme_reg_1 + send_data));
j = ilpeek((volatile long*)(vme_reg_1 + send_data));
j = ilpeek((volatile long*)(vme_reg_1 + send_data));

/* Set up middle module for pipeline */
printf("Setting up middle module...\n");
iwpoke((volatile short*)(vme_reg_2 + port_ctrl), LB_IDLE);
iwpoke((volatile short*)(vme_reg_2 + port_ctrl),
      MODE_PIPE | SEND_PORT_NONE | REC_PORT_NONE | LB_PIPE);
iwpoke((volatile short*)(vme_reg_2 + lb_reset), ZERO);
iwpoke((volatile short*)(vme_reg_2 + stop_cnt_msw), -1);
iwpoke((volatile short*)(vme_reg_2 + stop_cnt_lsw), -1);

/* Set up right-hand module to store data from VME */
printf("Setting up right-hand module...\n");
iwpoke((volatile short*)(vme_reg_3 + port_ctrl), LB_IDLE);
iwpoke((volatile short*)(vme_reg_3 + port_ctrl), SEND_PORT_NONE | REC_PORT_VME);

/* Clear out old data in right-hand module - not necessary, but
   do so to show new data arrived */
for (i = 1; i <= mem_loop; i += 4) {
    ilpoke((volatile long*)(vme_reg_3 + rec_data), -1);
    ilpoke((volatile long*)(vme_reg_3 + rec_data), -1);
    ilpoke((volatile long*)(vme_reg_3 + rec_data), -1);
    ilpoke((volatile long*)(vme_reg_3 + rec_data), -1);
}

/* Flush last word to memory, set up for consume */
ilpoke((volatile long*)(vme_reg_3 + rec_data), ZERO);
iwpoke((volatile short*)(vme_reg_3 + port_ctrl), LB_IDLE);
```

Continued on Next Page

```
iwpoke((volatile short*)(vme_reg_3 + port_ctrl),
       MODE_CONSUME | SEND_PORT_NONE | REC_PORT_LBUS);
iwpoke((volatile short*)(vme_reg_3 + lb_reset), ZERO);

/* Set up stop address */
i = mem_amt - REC_OFFSET;
iwpoke((volatile short*)(vme_reg_3 + stop_cnt_msw), (short)(i >> 16));
iwpoke((volatile short*)(vme_reg_3 + stop_cnt_lsw), (short)i);

/* Initiate modules */
printf("Initiating modules...\n");
iwpoke((volatile short*)(vme_reg_3 + port_ctrl),
       iwpeek((volatile short*)(vme_reg_3 + port_ctrl)) | LB_ENABLE);
iwpoke((volatile short*)(vme_reg_2 + port_ctrl),
       iwpeek((volatile short*)(vme_reg_2 + port_ctrl)) | LB_ENABLE);
iwpoke((volatile short*)(vme_reg_1 + port_ctrl),
       iwpeek((volatile short*)(vme_reg_1 + port_ctrl)) | LB_ENABLE);

/* Wait for completion */
printf("Waiting for completion...\n");
i = 200000L;
do {
    temp = iwpeek((volatile short*)(vme_reg_1 + status));
} while (!(temp & LB_PAUSED) && —i);

/* Verify modules paused */
temp = iwpeek((volatile short*)(vme_reg_1 + status));
if (!(temp & LB_PAUSED)) {
    printf(">>>ERROR: Left-hand module not paused\n");
    exit(1);
}
temp = iwpeek((volatile short*)(vme_reg_2 + status));
if (!(temp & LB_PAUSED)) {
    printf(">>>ERROR: Middle module not paused\n");
    exit(1);
}
temp = iwpeek((volatile short*)(vme_reg_3 + status));
if (!(temp & LB_PAUSED)) {
    printf(">>>ERROR: Right-hand module not paused\n");
    iwpoke((volatile short*)(vme_reg_3 + lb_flush), ZERO);
    iwpoke((volatile short*)(vme_reg_3 + lb_reset), ZERO);
    exit(1);
}
```

Continued on Next Page

```
    /* Flush out last word from Controller chip; read amount of data received */
    iwpoke((volatile short*)(vme_reg_3 + lb_flush), ZERO);
    i = (long)(iwpeek((volatile short*)(vme_reg_3 + curr_addr_msw)) << 16) |
            (long)iwpeek((volatile short*)(vme_reg_3 + curr_addr_lsw)) - 3;
    /* Flush out last word to memory.  Note: must use iwpeek here vs. iwpoke
       above since have SEND_PORT_LBUS set.  A write here would cause driver
       contention on the memory bus */
    temp = ilpeek((volatile long*)(vme_reg_3 + send_data));

    printf("Verifying data amount received...\n");
    if (i != mem_amt) {
        printf(">>>ERROR: Wrong amount of data received\n");
        printf("  Expected %ld\n", mem_amt);
        printf("  Got %ld\n", i);
        exit(1);
    }

    /* Set up right-hand module to read data via VME */
    iwpoke((volatile short*)(vme_reg_3 + port_ctrl), LB_IDLE);
    iwpoke((volatile short*)(vme_reg_3 + port_ctrl), SEND_PORT_VME | REC_PORT_NONE);
    j = ilpeek((volatile long*)(vme_reg_3 + send_data));
    j = ilpeek((volatile long*)(vme_reg_3 + send_data));
    j = ilpeek((volatile long*)(vme_reg_3 + send_data));
    j = ilpeek((volatile long*)(vme_reg_3 + send_data));
    printf("Comparing data received...\n");

    /* Verify data in right-hand module */
    for (i = 0; ++i <= mem_loop;) {
        if ((j = ilpeek((volatile long*)(vme_reg_3 + send_data))) != i) {
            printf(">>>ERROR: Data error at location %ld\n", i);
            printf("  Expected %ld\n", i);
            printf("  Got %ld\n", j);
        }
    }
    exit(0);
}
```

Index

A

Abbreviated commands, 80
ABORt, 48, 83
ABORt Subsystem
    ABORt, 83
Aborting data transfer prematurely, 48
Address, memory module logical, 13
Address, setting the logical, 28
Address, the base, 133
Addressing the memory module (External controller, 13
Addressing the memory module (External controller and PC), 13
Addressing, register, 131
Allocating memory, 40
Allowable signal levels, 70
Assignments, register, 134

B

Back up method
    enabling, 29
    selecting, 29
Bus, how data is transferring on the Local, 48
Bus, outputting data to the Local, 45
Bus, storing data to the Local, 41
Bus, transferring data using the Local, 41
Bus, using multiple memory modules on the Local, 49

C

C compiler used, 15
C language example programs, 15
C program example, 16
*CLS, 110
Cautions, 27
Certification, 5
Changing IBASIC programs to BASIC programs, 20
Characteristics, general VXI, 126
Codes, error, 127 - 130
Command module requirements, 12
Command reference, 79
Command types, 79
Commands
    VINStrument[:CONFigure]:LBUSp, 41 - 42, 45, 49 - 50, 54 - 55, 60 - 61
    ABORt, 48, 83
    DISPlay:MONitor[:STATe], 84
    FORMat[:DATA], 35, 85
    IEEE-488.2 common, 109 - 118
    INITiate:CONTinuous, 86
    INITiate[:IMMediate], 42, 45 - 46, 50 - 51, 56, 61
    INITiate[:IMMediate] , 87
    MEMory:FREE?, 88
    MEMory:NONVolatile?, 88
    STATus:OPC:INITiate, 42, 45, 50, 55, 60 - 61, 89
    STATus:OPERation|QUEStionable:CONDition?, 90
    STATus:OPERation|QUEStionable:ENABle, 90
    STATus:OPERation|QUEStionable:NTRansition, 92
    STATus:OPERation|QUEStionable:PTRansition, 92
    STATus:OPERation|QUEStionable[:EVENt]?, 91
    STATus:PRESet, 93
    SYSTem:ERROR?, 94
    SYSTem:VERSion?, 94 - 95
    TRACe|DATA:CATalog?, 96
    TRACe|DATA:DATA?, 39
    TRACe|DATA:DEFine, 34, 40, 48, 98
    TRACe|DATA:DELete:ALL, 34, 42, 54 - 55, 99
    TRACe|DATA:DELete[:NAME], 100
    TRACe|DATA[:DATA], 34, 96
    TRACe|DATA[:DATA]:COUNt, 97
    TRACE|DATA[:DATA]:COUNt?, 35
    TRACe|DATA[:DATA]:PARTial?, 35, 98
    VINStrument:IDENtity?, 108
    VINStrument[:CONFigure]:LBUS:PAUSed?, 103
    VINStrument[:CONFigure]:LBUS:RESet, 41 - 42, 45, 49 - 50, 54 - 55, 60
    VINStrument[:CONFigure]:LBUS:RESet , 104
    VINStrument[:CONFigure]:LBUS[:MODE], 101 - 102
    VINStrument[:CONFigure]:TEST:DATA, 104
    VINStrument[:CONFigure]:TEST:DATA?, 105
    VINStrument[:CONFigure]:VME:RECeive:ADDRess:Data?, 107
    VINStrument[:CONFigure]:VME:RECeive:ADDRess:READy?, 107
    VINStrument[:CONFigure]:VME:SEND:ADDRess:DATA?, 107

VINStrument[:CONFigure]:VME:SEND:ADDRess:READy?, 108
VINStrument[:CONFigure]:VME[:MODE], 34, 41, 45, 49 - 50, 54 - 55, 60, 106
Commands, abbreviated, 80
Commands, implied, 80
Commands, linking, 82
Commands, linking IEEE 488.2 common commands with SCPI, 82
Common capabilities register, 144
Common command format, 79
Common Commands
*CLS, 110
*DMC, 110
*EMC, 111
*EMC?, 111
*ESE, 111
*ESE?, 111
*ESR?, 112
*GMC?, 112
*IDN?, 112
*LMC, 113
*LRN?, 113
*OPC, 114
*OPC?, 114
*PMC, 114
*RCL, 115
*RMC, 115
*RST, 116
*SAV, 116
*SRE, 117
*SRE?, 117
*STB?, 117
*TST?, 118
*WAI, 118
Compound command separator, 79
Configuration and initiation order for multiple modules, 64
Configuration, system, 15, 20, 23, 148
Configuring the Memory Module, 27
Configuring the memory size, 29
Conformity, declaration, 7
Connected MS-DOS® computer
externally, 13
using, 13
Controller, using an embedded, 13
Controller, using BASIC with an externally connected GPIB, 12
Current address MSW/LSW registers, 147

D

Data output from a single module to multiple memory modules, 64
Data output from multiple memory modules to a single module, 65
Data, how to transfer, 70
Data, outputting, 35
Data, storing, 34
Data, using more than two modules to receive, 65
Declaration of conformity, 7
Description register, 144
Description, Local Bus, 69
Description, module, 33
Descriptions, register, 135
Device driver, 11
Device type register, 135
Different ways to output data with multiple modules, 65
DISPlay
:MONitor[:STATe], 84
DISPlay Subsystem, 84
*DMC, 110
Documentation history, 6
Down, retrieving data after a power, 38
Driver, device, 11
Drivers, installing the, 12
Drivers, recommended ways to download the, 12

E

E1406 command module
using Agilent IBASIC in an Agilent E1405, 12
*EMC, 111
*EMC?, 111
Enabling and selecting the back up method, 29
Environment, using the QuickBASIC, 23
Error codes, 127 - 130
*ESE, 111
*ESE?, 111
*ESR?, 112
Example programs, 33
Example, C program, 16
Example, IBASIC program, 20
Example, program, 148
Example, QuickBASIC program, 24

F

FORMat
[:DATA], 85
FORMat Susbsystem, 85

Format, common command, 79
Format, SCPI command, 15, 79
FORMat[:DATA], 35, 39 - 40
Formats, returned data, 39

G

General information, 131
General VXI characteristics, 126
Getting started, 11
*GMC?, 112
Group, the operation status, 75
Group, the questionable data, 77
Group, the standard event, 74
Group, the status byte, 77

H

How data is transferring on the Local Bus, 48
How to transfer data, 70
Agilent E1488A module command quick reference, 119
Agilent IBASIC/BASIC language program, 20
GPIB, transferring data using, 34

I

IBASIC program example, 20
ID register, 135
*IDN?, 112
IEEE-488.2 common commands, 109 - 118
Implied commands, 80
Information, general, 131
Information, SCPI conformance, 123 - 124
INITiate
    :CONTinuous, 86
    [:IMMediate] , 87
INITiate Subsystem, 86 - 87
INITiate[:IMMediate], 42, 45 - 46, 50 - 51, 56, 61
Installing the Drivers, 12
Installing the memory module, 11
Instrument and programming languages, 14
Instrument specifications, 125
IRQ control register, 136

K

Key, Local Bus, 70

L

Languages
    instrument, 14
    programming, 14
Level, linking multiple SCPI commands at the same hierarchical, 82
Levels, allowable signal, 70
Line, using the command, 23
Lines, Local Bus data/control, 69
Linking commands, 82
Linking IEEE 488.2 common commands with SCPI commands, 82
Linking multiple SCPI commands at the same hierarchical level, 82
LInking multiple SCPI commands from different subsystems, 82
*LMC, 113
Loading programs, 20
Local bus data transfers, 69
Local bus data/control lines, 69
Local bus description, 69
Local bus flush register, 139
Local bus key, 70
Local bus operating modes, 71
Local bus reset register, 138
Local bus restart register, 139
Local Bus, transferring data using the, 41
*LRN?, 113

M

MEMory
    :FREE?, 88
    :NONVolatile?, 88
Memory attribute register, 137
Memory module logical address, 13
Memory module registers, 134
MEMory Subsystem, 88
Memory, allocating, 40
Memory, returning all data that is in, 39
Modes, Local Bus operating, 71
Module
    configuring the memory, 27
    understanding the memory, 69
    using the memory, 33
Module description, 33
Module, data output from multiple memory modules to a single, 65
Module, installing the memory, 11
Modules
    configuration and initiation order , 64
    data output from a single module to multiple memory, 64
    multiple, 64
Modules, different ways to output data with multiple, 65

Modules, outputting blocks of data in reverse using multiple memory, 67
Modules, outputting data as one block using more than two memory, 66
Modules, Outputting data as one block using two memory, 66
Modules, outputting data as separate blocks using two, 65
Modules, outputting data from multiple, 60
Modules, passing data through memory, 49
Modules, resetting order of the Local Bus cir, 48
Multiple, storing data into multiple , 54

O

*OPC, 114
*OPC?, 114
Or returned data
    order , 38
    stored, 38
Order for stored or returned data, 38
Outputting blocks of data in reverse using multiple memory modules, 67
Outputting data, 35
Outputting data as one block using more than two memory modules, 66
Outputting data as one block using two memory modules, 66
Outputting data as separate blocks using more than two, 66
Outputting data as separate blocks using two modules, 65
Outputting data from multiple modules, 60
Outputting data to the Local Bus, 45
Overview, status system, 73

P

Passing data through memory modules, 49
PC), 13
*PMC, 114
Port control register, 139
Prematurely, aborting data transfer, 48
Preparation for use, 11
Priority, selecting the interrupt, 32
Program example, 148
Program, Agilent IBASIC/BASIC language, 20
Program, register-based example, 148
Program, running a, 16, 23
Program, running the, 148
Programming
    register-based, 131
Programming, SCPI, 14
Programs, C language example, 15
Programs, changing IBASIC programs to BASIC, 20
Programs, example, 33
Programs, loading, 20
Programs, QuickBASIC language, 23
Purpose of the trace subsystem, 38

Q

QuickBASIC language programs, 23
QuickBASIC program example, 24

R

*RCL, 115
Receive status register, 146
Recommended ways to download the drivers, 12
Reference
    command, 79
Reference, Agilent E1488A module command quick, 119
Reference, SCPI command, 82
Register
    description, 144
    device type, 135
    IRQ control, 136
    local bus flush, 139
    local bus reset, 138
    memory attribute, 137
    port control, 139
    receive status, 146
    send status, 145
    subclass, 145
    VXI control, 136
    VXI status, 135
    common capabilities, 144
    ID, 135
    local bus restart, 139
Register addressing, 131
Register assignments, 134
Register descriptions, 135
Register-Based example program, 148
Register-Based Programming, 131
Registers, current address MSW/LSW, 147
Registers, memory module, 134
Registers, send/receive data MSW/LSW, 145
Registers, stop count MSW/LSW, 147
Requirements, command module, 12
Resetting order of the Local Bus circuitry of, 48
Retrieving data after a power down, 38
Returned data formats, 39
Returning all data that is in memory, 39
*RMC, 115

*RST, 116
Running a program, 16, 23
Running the program, 148

S

Safety warnings, 6
*SAV, 116
SCPI command format, 15, 79
SCPI command reference, 82
SCPI command structure, 14
SCPI conformance information, 123 - 124
SCPI programming, 14
Selecting the interrupt priority, 32
Send status register, 145
Send/Receive data MSW/LSW registers, 145
Separator, compound command, 79
Setting the logical address, 28
Setting the module configuration switches, 29
Size, configuring the memory, 29
Specifications, 125
Specifications, instrument, 125
*SRE, 117
*SRE?, 117
STATus
:OPC:INITiate, 89
:OPERation|QUEStionable:CONDition?, 90
:OPERation|QUEStionable:ENABle, 90
:OPERation|QUEStionable:NTRansition, 92
:OPERation|QUEStionable:PTRansition, 92
:OPERation|QUEStionable[:EVENt]?, 91
:PRESet, 93
STATus Subsystem, 89 - 93
Status system overview, 73
STATus:OPC:INITiate, 42, 45, 50, 55, 60 - 61
*STB?, 117
Stop count MSW/LSW registers, 147
Storing data, 34
Storing data into multiple modules, 54
Storing data to the Local Bus, 41
Structure, SCPI command, 14
Subclass register, 145
Subsystem, purpose of the trace, 38
Subsystems, LInking multiple SCPI commands from different, 82
Switches, setting the module configuration, 29
SYSTem
:ERROR?, 94
:VERSion?, 94 - 95
System configuration, 15, 20, 23, 148
SYSTem Subsystem, 94 - 95
System, the SCPI status, 73

T

The base address, 133
The operation status group, 75
The questionable data group, 77
The SCPI status system, 73
The standard event group, 74
The status byte group, 77
TRACe|DATA
:CATalog?, 96
[:DATA], 96
[:DATA]:COUNt, 97
[:DATA]:PARTial?, 98
:DEFine, 98
:DELete:ALL, 99
:DELete[:NAME], 100
TRACe|DATA Subsystem, 96 - 100
TRACe|DATA:DATA?, 39
TRACe|DATA:DEFine, 34, 40, 48
TRACe|DATA:DELete:ALL, 34, 42, 54 - 55
TRACe|DATA[:DATA], 34
TRACE|DATA[:DATA]:COUNt?, 35
TRACe|DATA[:DATA]:PARTial?, 35
Transferring data using GPIB, 34
Transferring data using the Local Bus, 41
Transfers, Local Bus data, 69
*TST?, 118
Two, Outputting data as separate blocks using more than, 66
Types, command, 79

U

Understanding the memory module, 69
Use, preparation for, 11
Used, C compiler, 15
Using an embedded controller, 13
Using and externally connected MS-DOS® computer, 13
Using BASIC with an externally connected GPIB controller, 12
Using Agilent IBASIC in an Agilent E1405 or Agilent 1406 command module, 12
Using more than two modules to receive data, 65
Using multiple memory modules on the Local Bus, 49
Using the command line, 23
Using the memory module, 33
Using the QuickBASIC environment, 23

V

VINStrument
[:CONFigure]:LBUS[:MODE], 101 - 102
[:CONFigure]:LBUS:PAUSed?, 103
[:CONFigure]:LBUS:RESet, 104
[:CONFigure]:TEST:DATA, 104
[:CONFigure]:TEST:DATA?, 105
[:CONFigure]:VME[:MODE], 106
[:CONFigure]:VME:SEND:ADDRess:DATA?, 107
[:CONFigure]:VME:SEND:ADDRess:READy?, 108
:IDENtity?, 108
VINStrument Subsystem, 101 - 108
VINStrument[:CONFigure]:LBUS:RESet, 41 - 42, 45, 49 - 50, 54 - 55, 60
VINStrument[:CONFigure]:LBUS[:MODE], 41 - 42, 45, 49 - 50, 54 - 55, 60 - 61
VINStrument[:CONFigure]:VME:RECeive:ADDRess:Data?, 107
VINStrument[:CONFigure]:VME:RECeive:ADDRess:READy?, 107
VINStrument[:CONFigure]:VME[:MODE], 34, 41, 45, 49 - 50, 54 - 55, 60
VXI control register, 136
VXI status register, 135

W

*WAI, 118
WARNINGS, 6, 27
Warnings and cautions, 27
Warranty, 5